Symbolic Weak Equivalences: Extension, Algorithms, and Minimization - Extended version

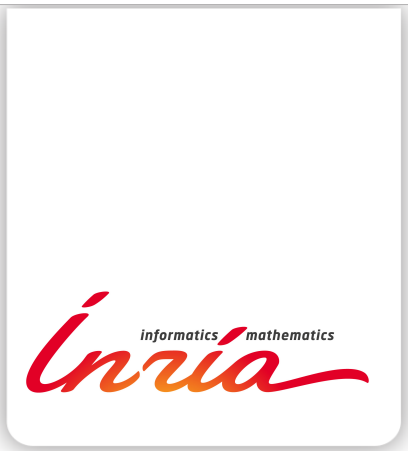
Biyang Wang, Eric Madelaine, Min Zhang

Symbolic Weak Equivalences: Extension, Algorithms, and Minimization - Extended version

Biyang WANG, Eric MADELAINE , Min ZHANG

RESEARCH REPORT
N° 9389
January 2021
Common Project-Team Kairos

ISSN 0249-6399 ISRN INRIA/RR--9389--FR+ENG



Symbolic Weak Equivalences: Extension, Algorithms, and Minimization - Extended version

Biyang WANG *, Eric MADELAINE †, Min ZHANG *

Common Project-Team Kairos

Research Report n° 9389 — January 2021 — 71 pages

Abstract: *Open Automata* (OA) are symbolic and parameterized models for open concurrent systems. Here open means partially specified systems, that can be instantiated or assembled to build bigger systems. An important property for such systems is "compositionality", meaning that logical properties, and equivalences, can be checked locally, and will be preserved by composition. In previous work, a notion of equivalence named FH-Bisimulation was defined for open-automata, coming with both *Strong* and *Weak* flavors, where *Weak* means ignoring internal moves when they have no effect on the external behavior. Both flavors have been proved to be congruences for the OA's composition. In this paper, we propose a new definition of (weak) open automata, that is both more expressive for encoding the behavior of parameterised systems, and suitable as a finite encoding of weak OAs. We name these *meta* open automata (meta-WOA), and provide two algorithms to check their equivalence, either explicitely building the meta-WOAs, or constructing their meta open transitions on-demand. The last strategy has better termination properties. Then we provide pattern-based Reduction rules for OA, and we discuss preservation of *Weak FH-Bisimulation* by such reductions.

Key-words: Concurrent Systems, Symbolic Bisimulation, Weak Bisimulation, Open Systems, SMT Solver, Reduction, Minimization

KAIROS is a common team between the I3S Lab (Univ. Côte d'Azur and CNRS) and INRIA Sophia Antipolis Méditerranée

* Shanghai Key Laboratory of Trustworthy Computing, Software College, East China Normal University, Shanghai, China

† INRIA/I3S Kairos; Université Côte d'Azur, Sophia Antipolis, France

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Équivalences faibles symboliques : Extension, Algorithmes et Minimisations - Version étendue

Résumé : Les *Automates Ouverts* (OA) sont des modèles symboliques et paramétrés pour les systèmes concurrents ouverts. Ici, ouvert signifie des systèmes partiellement spécifiés, qui peuvent être instanciés ou assemblés pour construire des systèmes plus grands. Une propriété importante pour de tels systèmes est la «compositionnalité», ce qui signifie que les propriétés logiques et les équivalences peuvent être vérifiées localement et seront préservées par composition. Dans les travaux précédents, une notion d'équivalence nommée FH-Bisimulation a été définie pour les automates ouverts, en deux versions *Forte* et *Faible*, où *Faible* signifie ignorer les mouvements internes des composants du systéme lorsqu'ils n'ont aucun effet sur le comportement externe. Les deux versions se sont avérées être des congruences pour la composition des OAs. Dans cet article, nous proposons une nouvelle définition des automates ouverts (faibles), qui est plus expressive pour coder le comportement des systèmes paramétrés, et qui permet aussi un codage fini des OAs faibles. Nous les nommons *meta* automates ouverts (meta-WOA), et fournissons deux algorithmes pour vérifier leur équivalence, soit en construisant explicitement les méta-WOA, soit en construisant leurs méta-transitions ouvertes à la demande. La dernière stratégie a de meilleures propriétés de terminaison. Ensuite, nous fournissons des règles de réécriture permettant de réduire la taille des OAs, et nous discutons de la préservation de la bisimulation faible par ce type de réductions.

Mots-clés : Systèmes concurrents, Bisimulation Symbolique, Bisimulation Faible, Systèmes ouverts, Solveur SMT, Réduction, Minimisation

1 Introduction

Process algebras from [2] is a kind of mathematically rigorous language that can be used to describe and verify properties of concurrent communicating systems. *Operational semantics*, associated with each construct of a process algebra, is one of the most successful techniques for describing the formal, specific behavior of concurrent systems. It can model concurrent programs or systems as *labeled transition systems* (LTSs) that consist of a set of states, transition labels, and relations. Based on those contexts, behavioral equivalences have been developed as three kinds of equivalences known as trace equivalences, decorated-trace equivalences, and bisimulation-based equivalences. [21] and [4] reviewed more of these equivalences.

In the nineties, reasearch by [5, 14, 6, 15] extended the basic behavioral models based on LTSs to address value-passing or parameterized systems, using various symbolic encodings of the transitions. [15] addressed value-passing calculi, for which he developed a symbolic behavioral semantics and proved algebraic properties. Separately [8] defined another symbolic semantics for a parameterized broadcast calculus, together with strong and weak bisimulation equivalences, developed a symbolic model-checker based on a tableau method for these processes. Later, [11] gives a survey of semantic theory for value-passing processes, focusing on bisimulation equivalences with early or late semantics.

Minimization (Reduction) of automata is another topic but also relates to symbolism and bisimulation. [3] formally defined and proved the basic properties of minimality in the symbolic setting, and lift classical minimization algorithms, Huffman-Moore's and Hopcroft's algorithms, to symbolic automata. [16] designed three on-the-fly reductions of LTSs: τ -compression, τ-closure , and τ-confluence. These three reduction operations preserve weak bisimulation between the original and reduced systems.

[9] offers a methodology using open, symbolic, and parameterized models, endowed with a notion of symbolic bisimulation. It defines a new behavioral specification formalism called 'Open parameterized Networks of Synchronized Automata (pNet)' for distributed, synchronous, asynchronous, or heterogeneous systems. The pNet model has a hierarchical and tree-like structure that gives it a strong ability to describe and compose complex systems. It is symbolic in the sense that it explicitly manipulates data types and data-expressions as arguments to communication and synchronisation. And it is open thanks to a notion of *Holes*, representing unspecified subsystems, that can be instantiated, providing a powerful methodology for compositional specification and verification activities. The symbolic and open aspects of pNets give them the potential to use small state space to represent large systems.

PNets were given a behavioral semantics in terms of open automata, that we will use an extend in this work. Bisimulation equivalences over open automata are symbolic as they manipulate data expressions rather than individual values, and come as an extension of notions introduced in [11]. In previous works, it was shown that these equivalences are preserved by instantiation and composition of PNets, and how to devise algorithms to check them using an SMT engine to manage the reasoning on the data part.

This article provides several new definitions, enriching the theory about parameterized networks of synchronized automata and open automata, especially for weak open automata. These definitions can help the semantic analysis of weak bisimulation. We design an on-the-fly method with two kinds of searching styles for checking whether a given relation is a weak bisimulation. We also offer a minimization method, One State Minimization, which minimizes any OA into an equivalent one with only one state. Besides that, we propose three rewriting rules for the reduction of open automata. To relate formally *local* rewriting with the global properties of the system, we define and prove a *Local Bisimulation* principle, that can be used to prove that reduction rules preserve weak bisimulation. Finally we show our reduction rules are effective on

our running example.

1.1 Motivation

Formal methods, including all kinds of automaton, simulation equivalence, are widely studied and used in academia and industry. Automata is a modeling tool that helps the designer focus on the states of variables or the actions the system can do under certain situations. Bisimulation is a kind of equivalence between two systems with bidirectional simulation. This equivalence guarantees that the actions the system can perform current and future can be simulated by another system, and vise versa. In other words, the two systems are indistinguishable from the outer observers (users) or cooperation systems.

For example, a modular system with several components and significant properties has been proven with high cost. It is now required to upgrade one of its components without changing the interfaces but new structure details for shorter response delay or more accurate results. With the help of automata modeling, if the equivalence of bisimulation between the new sub-part and the original one is proven, then there is no need to prove properties for the new system (the one with the new sub-part) again because the whole system keeps its behavioral properties. It is so-called composition properties.

Strong bisimulation requires the same actions from the two systems, which is too strict for practice. Weak bisimulation can tolerate several internal actions with only one observable action among them, which is more commonly used in practice. In software development, some interface upgrading requires the weak bisimulation relation, and the interfaces keep the same but without constraints on the internal structure and actions.

1.2 Previous Works

Open *Parameterized Networks of Synchronized Automata* (pNets) [9] are semantic objects used for modeling the behavior of concurrent open systems, where 'Open' means partially specified as systems with 'Holes'. In the previous work, we have already defined operational semantics of open pNets for constructing Open Transitions (OT) and transforming open pNets to equivalent *Open Automata* (OA)[20].

For these synchronized internal actions in sub-level of pNet which cannot be observed in the external environment, as known as τ in [18], we defined the construction rules for *Weak Open Transition* (WOT), ignoring any number of transitions with non-observable actions, and defined *Weak Open Automata* (WOA).

To compare the behavior capabilities between OAs, notions of equivalences named *FH-Bisimulation* (Formal Hypothesis) [9] and *StrFH-Bisimulation* (Structured FH) [10] have been defined for intrinsically infinite open automata and strong bisimulation between open automata with finite encoding, respectively. These bisimulations are preserved by pNets composition, and this property make them essential for our compositional verification methodology. Extending these two definitions, we have defined the *Weak FH-Bisimulation* for both OAs and WOAs and we have shown under which conditions this new equivalence is a for pNets composition (work currently under submission: [1]).

1.3 Contributions.

Here are the main contributions in this paper:

Contribution 1: We extend the definition of Open Automaton to include a new kind of weak open transition, *meta-WOT*, which uses meta-Variables to complete their semantic meaning.

The meta-Variables provide scalability for transitions and systems; In this work, they are used to encode in a finite way internal loops of the automata, allowing an algorithmic handling of weak bisimulation. But they could also be used at early design time, to enable finite description of systems with parametric topology, like arrays of processes. We enrich the theory of weak bisimulation to include meta WOTs.

Contribution 2: We analyze two ways of checking whether a given relation is a weak bisimulation, by (1) building completed meta-WOAs before checking relations and (2) searching for expected WOT on-the-fly while checking relations. We argue that the second way is more efficient and easy to implement, and provide an algorithm.

Contribution 3: We discuss weak-bisimulation preserving minimization methods, and show why it is not useful in practice. Then propose three pattern-based rewriting rules for reducing a given open automaton, and show their effectiveness on a small case-study. We prove one of the rules preserves bisimulation relation.

1.3.1 Structure.

This article is organized as follows. Section 2 provides the basic definitions of pNets, OT, OA, bisimulations, introducing our running example, and all other existing notations as the basis. Section 3 defines the meta-WOT, meta-WOA, meta FH-Bisimulation, and other new conceptions we define. We also analyze two different methods for checking weak bisimulation here. Section 4 presents the checking algorithm. Section 5 gives an analysis of the minimization method, defines our three reduction rules and proves their preservation properties. Section 6 is about related works. Finally, Section 7 concludes the paper. Finally, the appendices show:

- the details of the weak transition of the Implementation Automaton, and some more details of the weak bisimulation checking of our running example.
- A new proposal for defining meta-OTs and meta-OAs, and the associated bisimulation notion and checking algorithm.
- Three additional examples specifications.

2 Background and notations

This section introduces the notations we will use in this article and recalls the definition of Open Transitions, Open Automata, their Weak versions, and the corresponding bisimulations equivalences that were first defined in [9, 20, 10, 1]. A significant difference with the definitions in the first three references, essential for building weak open transitions, is that one of the restrictions from previous papers, stating variables should be local to a state in LTSs, was removed in the most recent work [1], providing a better foundation for weak automata theory.

2.1 Notations

2.1.1 Term algebra

Formally, we assume the existence of a term algebra $\mathbb{T}$, where Σ is the signature of the data and action constructors. Within $\mathbb{T}$, we distinguish a set of expressions $\mathbb{E}$, including a set of boolean expressions $\mathbb{B}$ ($\mathbb{B} \subseteq \mathbb{E}$). We let e_i range over expressions ($e_i \in \mathbb{E}$). On top of $\mathbb{E}$ we build the action algebra $\mathbb{A}$, with $\mathbb{A} \subseteq \mathbb{T}, \mathbb{E} \cap \mathbb{A} = \emptyset$; naturally, action terms will use data expressions as subterms. The function $vars(t)$ identifies the set of variables in a term $t \in \mathbb{T}$.

We let e_i range over expressions ($e_i \in \mathbb{E}$), a range over action labels, *op* be operators, and x_i and y_i range over variable names. More precisely, we consider 2 sets of variables, *standard* variables that are used in expressions, and *input* variables that appear only in parameters of communication actions. In the following grammar of action expressions, parameters are input variables or data expressions:

$$\begin{array}{rcll} \alpha \in \mathbb{A} & ::= & a(p_1, \ldots, p_n) & \text{action term} \\ p_i & ::= & ?x \mid e_i & \text{parameter} \\ e_i & ::= & \mathit{Value} \mid x \mid op(e_1, .., e_n) & \text{expression} \end{array}$$

$\uplus$ is the disjoint union on sets. We extend it to a disjoint union of indexed sets defined by the merge of the two sets provided they are indexed on disjoint families. The elements of the union of two indexed sets are then accessed by using an index of one of the two joined families. An indexed family is denoted as follows: $a_i^{i \in I}$ is a family of elements a_i indexed over the set I.

2.1.2 Substitution

We denote $y \leftarrow e$ a substitution. The application of the substitution is denoted $\{\!\{y \leftarrow e\}\!\}$, the operation replaces in a term all occurrences of the variable y by the expression e. *Post* ranges over (indexed) sets of substitutions; $\{\!\{Post\}\!\}$ is the substitution that applies all the substitutions defined by *Post* in a parallel manner. $\otimes$ is the composition operator on substitutions, such that for any term t we have: $t\{\!\{Post \otimes Post'\}\!\} = (t\{\!\{Post'\}\!\})\{\!\{Post\}\!\}$.

For this property to be valid, even if the substitution does not operate on all variables, we define the composition operation as follows:

$$(x_k \leftarrow e_k)^{k \in K} \otimes (x'_{k'} \leftarrow e'_{k'})^{k' \in K'} = (x_k \leftarrow e_k\{\!\{(x'_{k'} \leftarrow e'_{k'})^{k' \in K'}\}\!\})^{k \in K} \cup (x'_{k'} \leftarrow e'_{k'})^{k' \in K''}$$

where $K'' = \{k' \in K' | x'_{k'} \notin \{x_k\}^{k \in K}\}$.

2.2 PNets: Parameterised Networks of Synchronized Automata

pNets are tree-like structures, where the leaves are either *parameterised labeled transition systems*, expressing the the behavior of basic processes, or *Holes*, used as placeholders for unknown processes. Behaviors (parameterised actions) of each component (sub-pNets, pLTS, or Holes) on the same level are synchronized by using a set of *synchronisation vectors*.

2.2.1 Parameterised Labelled Transition System (pLTS)

is a labeled transition system with variables; variables can be used in actions, guards, and assignments.

Definition 2.1 (pLTS) *A pLTS is a tuple* $pLTS \triangleq \langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle$ *where:*

- S *is a set of states.*
- $s_0 \in S$ *is the initial state.*
- V *is a set of global variables for the pLTS,*
- $\rightarrow \subseteq S \times L \times S$ *is the transition relation and* L *is the set of labels of the form:* $\langle \alpha,\ e_b,\ (x_j := e_j)^{j \in J} \rangle$*, where* $\alpha \in \mathbb{A}$ *is a parameterised action,* $e_b \in \mathbb{B}$ *is a guard, and the variables* x_j *are assigned the expressions* $e_j \in \mathbb{E}$*. If* $s \xrightarrow{\langle \alpha,\ e_b,\ (x_j := e_j)^{j \in J} \rangle} s' \in \rightarrow$ *then* $vars(\alpha) \backslash iv(\alpha) \subseteq V$*,* $vars(e_b) \subseteq vars(s) \cup vars(\alpha)$*, and* $\forall j \in J.\ (vars(e_j) \subseteq V \cup iv(\alpha) \wedge x_j \in V)$*.*

The transitions in pLTS are constructed by action labels (labeling the unique name of the action), guards (a predicate describes the occurrence condition of the transition), variables (the variables make it possible to represent infinity ground transitions with finite symbolic transitions), and parallel assignments of variables (so that their order do not matter and they all use the values of variables before the transition or the values received as action parameters).

Now we define pNet nodes as constructors for hierarchical behavioral structures. A pNet has a set of sub-pNets that can be either pNets, pLTSs, or a set of Holes, playing the role of process parameters. A pNet is thus a composition operator that can receive processes as parameters; it expresses how the sub-processes' actions synchronize. The structure of pNets is defined in Definition 2.2, which relies on the definition of Holes, leaves, and sorts formalized in Definition 2.3.

Definition 2.2 (pNets) *A pNet P is a hierarchical structure where leaves are pLTSs and Holes: $P \triangleq pLTS \mid \langle\!\langle P_i^{i\in I}, Sort_j^{j\in J}, SV_k^{k\in K}\rangle\!\rangle$ where*

- *$P_i^{i\in I}$ is the family of sub-pNets indexed over I. $vars(P_i)$ and $vars(P_j)$ must be disjoint for $i \neq j$.*
- *J is a set of indexes, called* Holes*. I and J are* disjoint*: $I \cap J = \emptyset$, $I \cup J \neq \emptyset$*
- *$Sort_j \subseteq \mathbb{A}_S$ is a set of action terms, denoting the* sort *of Hole j.*
- *$SV_k^{k\in K}$ is a set of synchronisation vectors. $\forall k \in K.\, SV_k = \alpha_l^{l\in I_k \uplus J_k} \rightarrow \alpha'_k[e_k]$ where $\alpha'_k \in \mathbb{A}_S$, $I_k \subseteq I$, $J_k \subseteq J$, $\forall i \in I_k.\, \alpha_i \in Sort(P_i)$, $\forall j \in J_k.\, \alpha_j \in Sort_j$, and $vars(\alpha'_k) \subseteq \bigcup_{l\in I_k\uplus J_k} vars(\alpha_l)$. The global action of a vector SV_k is α'_k. $e_k \in \mathbb{B}$ is a guard associated to the vector such that $vars(e_k) \subseteq \bigcup_{l\in I_k\uplus J_k} vars(\alpha_l)$.*

Synchronisation vectors are identified modulo renaming of variables that appear in their action terms.

With several proved procedures in previous work, we can transform a pNet to a semantically equivalent OA. However, it is trivial to go into the details of the transformation in this paper.

Definition 2.3 (Sorts, Holes, Leaves, Variables of pNets)

- *The sort of a pNet is its signature, i.e., the set of actions in $\mathbb{A}_S$ it can perform, where each action signature is an action label plus the arity of the action. In the definition of sorts, we do not need to distinguish input variables, and we remove the* input marker *(?) of variables.*

$$\begin{aligned}&\mathrm{Sort}(\langle\!\langle S, s_0, V, \rightarrow\rangle\!\rangle) = \{\alpha\{\!\{?x \leftarrow x | x \in iv(\alpha)\}\!\} | s \xrightarrow{\langle \alpha,\ e_b,\ (x_j := e_j)^{j\in J}\rangle} s' \in \rightarrow\}\\ &\mathrm{Sort}(\langle\!\langle \overline{P}, \overline{Sort}, \overline{SV}\rangle\!\rangle) = \{\alpha' | \overline{\alpha} \rightarrow \alpha'[e_b] \in \overline{SV}\}\end{aligned}$$

- *The set of variables of a pNet P, denoted $vars(P)$, is the disjoint union of the sets of variables of all pLTSs that compose P.*
- *The set of Holes* $\mathrm{Holes}(P)$ *of a pNet is the indexes of the Holes of the pNet itself plus the indexes of all the Holes of its sub-pNets. It is defined inductively (we suppose those index sets disjoint):*

$$\begin{aligned}&\mathrm{Holes}(\langle\!\langle S, s_0, V, \rightarrow\rangle\!\rangle) = \emptyset\\ &\mathrm{Holes}(\langle\!\langle P_i^{i\in I}, \overline{Sort}, \overline{SV}\rangle\!\rangle) = J \uplus \bigcup_{i\in I} \mathrm{Holes}(P_i)\\ &\forall i \in I.\ \mathrm{Holes}(P_i) \cap J = \emptyset\\ &\forall i_1, i_2 \in I.\, i_1 \neq i_2 \Rightarrow \mathrm{Holes}(P_{i_1}) \cap \mathrm{Holes}(P_{i_2}) = \emptyset\end{aligned}$$

The sort of a Hole is its signature, i.e., the set of actions it can perform, where each action signature is an action label plus the arity of the action.

- *The set of leaves of a pNet is the set of all pLTSs occurring in the structure, as an indexed family of the form* $\text{Leaves}(P) = \langle\!\langle P_i \rangle\!\rangle^{i \in L}$.

$$\text{Leaves}(\langle\!\langle S, s_0, V, \rightarrow \rangle\!\rangle) = \emptyset$$
$$\text{Leaves}(\langle\!\langle P_i^{i \in I}, \overline{Sort}, \overline{SV} \rangle\!\rangle) = \biguplus_{i \in I} \text{Leaves}(P_i) \uplus \{i \mapsto P_i | P_i \textit{ is a pLTS}\}$$

A pNet Q *is* closed *if it has no Hole:* $\text{Holes}(Q) = \emptyset$*; else it is said to be* open.

2.2.2 Running Example

To illustrate this work, we use a simple communication protocol that provides safe transport of data between two processes over unsafe media. Its Specification and Implementation ('Specification' and 'Implementation' denote the Running Example) are originally encoded by pNets (see [1]), from which we build the corresponding Open Automata.

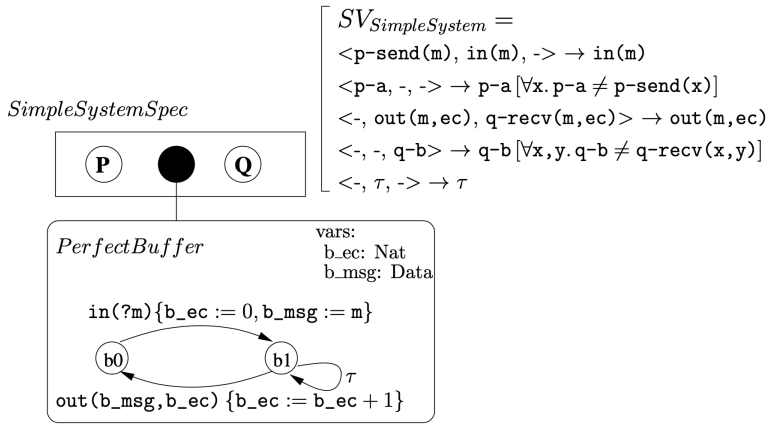


Figure 1: pNet of the protocol specification

To spare space, we only show and explain here the pNet of the Specification, in Figure 1. In the Specification, the pNet has only one level with two Holes (black box participant processes P and Q) and one pLTS, which describes the external behavior of the (perfect) protocol.

The actions of P, Q and pLTS (*PerfectBuffer*) are synchronized by synchronization vectors. Process P tries to communicate with Q by action `p-send(m)` with message data `m`. The protocol gets the message data by action `in(m)` with assignments `b_ec` $\leftarrow 0$, `b_msg` $\leftarrow$ `m`, where `b_ec` indicates the number of errors detected and `b_msg` represents the message payload. According to the last vector, the Buffer can do τ actions independently from any P's or Q's moves. This internal move models the capability of the protocol to detect errors: `b_ec` can increase by one unit at any time by a τ action. That is also why we can refine this system by its implementation. Finally, the protocol system passes the message and error count by action `out(b_m,b_ec)` and Q receives them by `q-recv(b_m,b_ec)`. From the second and fourth vectors, P and Q can do any action they want except the meaningful actions (`p-send` and `q-recv`) of the protocol.

2.3 Open Transitions and Open Automata

Open automata (OA) are not composition structures but are made of transitions that are dependent on the Holes' actions, and they can reason on a set of variables (potentially with only symbolic values). We use Open Automata as the semantic model for the behaviour of pNets (transformed by semantic operation) in [9]'s work.

Definition 2.4 (Open Transition) *An* open transition *(OT) over a set J of Holes with sorts $Sort_j^{j\in J}$, a set V of variables, and a set of states $\mathcal{S}$ is a structure of the form:*

$$\frac{\beta_j^{j\in J'}, Pred, Post}{s \xrightarrow{\alpha} s'}$$

Where $J' \subseteq J$, $s, s' \in \mathcal{S}$ and β_j is a transition of the Hole j, with $\beta_j \in Sort_j$. α is an action label denoting the resulting action of this open transition. Pred is a predicate over the variables in V and all variables in the different terms β_j and α. Post is a set of assignments that are effective after the open transition*, they are represented as a substitution of the form $(x_k \leftarrow e_k)^{k\in K}$ where $\forall k.\, x_k \in V$, and e_k are expressions over the variables V and all variables in the different terms β_j and α. Because the variables in V can be on both sides of the assignments, the assignments are always applied in a parallel manner. Open transitions are identified modulo logical equivalence on their predicate. We define $vars(ot) = vars(\alpha) \cup vars(Post) \cup vars(Pred) \cup^{j\in J} vars(\beta_j)$ the set of all standard variables in ot, and $otvars(ot)$ its input variables, that are local to the transition (and cannot occur on the left-hand side of an assignment).*

The Hole ($\beta_j^{j\in J'}$) is some sub-system that the designer only know its specification (external actions) but ignore its internal structures. That's why the transition is called 'Open'. At the pNet level, these Holes can be replaced by certain well-designed sub-pNets or pLTS. Of course, it will affect the structure of the whole OA. Sort of a Hole (or pNet, pLTS) is the action it can perform, defined as the signature in our previous work.

2.3.1 Open Automaton

An open automaton is an automaton where each transition is an open transition.

Definition 2.5 (Open automaton) *An* open automaton *is a structure of the form: $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$ where:*

- *J is a set of indices (Holes),*
- *$\mathcal{S}$ is a set of states and s_0 an initial state among $\mathcal{S}$,*
- *V is a set of variables of the automaton and each $v \in V$ may have an initial value $init(v)$.*
- *$\mathcal{T}$ is a set of open transitions and for each $t \in \mathcal{T}$ there exist J' with $J' \subseteq J$, such that t is an open transition over J' and $\mathcal{S}$.*

We take in this article a semantics and logical understanding of these automata. Open automata are closed by a simple form of refinement that allows us to refine the predicate or substitute any free variable by an expression. More formally, let Pred be any predicate and Post any substitution such that $V \cap \mathrm{dom}(Post) = \emptyset$. Then we have the following implication:

$$\frac{\overline{\beta}, Pred', Post'}{t \xrightarrow{\alpha} t'} \in \mathcal{T} \implies \frac{\overline{\beta}\{\!\{Post\}\!\}, Pred'\{\!\{Post\}\!\} \wedge Pred, Post \otimes Post'}{t \xrightarrow{\alpha\{\!\{Post\}\!\}} t'} \in \mathcal{T}$$

2.3.2 Weak Models

We first specify in terms of open transition, what it means for an action to be non-observable. Namely, we restrain ourselves to systems where the emission of a τ action by a sub-pNet cannot be observed by the surrounding pNets. In other words, a pNet cannot change its state or emit a specific observable action when one of its Holes emits a τ action.

More precisely, we state that τ is not observable if the automaton always allows any τ transition from Holes, and additionally, the global transition resulting from a τ action of a Hole is a τ transition not changing the pNet's state.

Definition 2.6 (Non-observability of τ actions) *Non-observability of τ actions for open automata. An open automaton $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$* cannot observe τ actions *if and only if for all j in J and s in $\mathcal{S}$ we have:*

1. $$\frac{(j \mapsto \tau), \mathit{True}, ()}{s \xrightarrow{\tau} s} \in \mathcal{T}$$

   and

2. *for all β_j, J, α, s, s', $Pred$, $Post$ such that*

   $$\frac{\beta_j^{j \in J}, \mathit{Pred}, \mathit{Post}}{s \xrightarrow{\alpha} s'} \in \mathcal{T}$$

   If there exists j such that $\beta_j = \tau$ then we have:

   $$\alpha = \tau \wedge \ s = s' \wedge \ \mathit{Pred} = \mathit{True} \wedge \mathit{Post} = () \wedge \ J = \{j\}$$

The first statement of the definition states that the open automaton must allow a Hole to do a silent action at any time, and must not observe it, i.e., it cannot change its internal state just for a Hole does a τ transition. The second statement ensures that there cannot be in the open automaton other transitions that would be able to observe a τ action from a Hole. The condition $J = \{j\}$ is a bit restrictive. It could safely be replaced by $\forall j \in J. \beta_j = \tau$, allowing the other Holes to perform τ transitions too (because these τ actions cannot be observed).

By definition, one weak open transition contains several open transitions, where each open transition can require an observable action from a given Hole. The same Hole might have to emit several observable actions for a single weak open transition to occur. Consequently, when a weak open transition is triggered, a sequence of actions from a given Hole may be required.

Thus, we let γ range over sequences of action terms and use $\uplus$ as the concatenation operator that appends sequences of action terms: given two sequences of action terms $\gamma \uplus \gamma'$ concatenates the two sequences. The operation is lifted to indexed sets of sequences: $\overline{\gamma_1} \uplus \overline{\gamma_2}$ is an indexed set such that, at each index i, $\overline{\gamma_1} \uplus \overline{\gamma_2}$ concatenates the sequences of actions at index i of $\overline{\gamma_1}$ and the one at index i of $\overline{\gamma_2}$[1]. $[a]$ denotes a sequence with a single element.

As required actions are now sequences of observable actions, we need an operator to build them from a set of actions that occur in open transitions, i.e., an operator that takes a set of actions performed by one Hole and produces a sequence of observable actions.

Thus we define $(\overline{\beta})^{\nabla}$ as the mapping $\overline{\beta}$ with only observable actions of the Holes in I, but where each element is either empty or a list of length 1:

$$(\beta_i^{i \in I})^{\nabla} = [\beta_i]^{i \in I'} \text{ where } I' = \{i | i \in I \wedge \beta_i \neq \tau\}$$

[1] One of the two sequences is empty when $i \notin \mathrm{dom}(\overline{\gamma_1})$ or $i \notin \mathrm{dom}(\overline{\gamma_2})$.

As an example the $(\overline{\beta})^{\nabla}$ built from the transition SI_3 in Figure 3 is $\texttt{P}\mapsto[\texttt{p-send(m)}]$. Remark that in our simple example no τ transition involves any visible action from a Hole, so we have no β sequences of length longer than 1 in the weak automaton.

Definition 2.7 (Weak open transition) *A weak open transition over a set J of Holes and a set of states $\mathcal{S}$ is a structure* $\dfrac{\gamma_j^{j\in J'}, Pred, Post}{s \overset{\alpha}{\Rightarrow} s'}$, *where $J' \subseteq J$, $s, s' \in \mathcal{S}$ and γ_j is a list of transitions of the Hole j, with each element of the list in $Sort_j$. V is a set of global variables. α is an action label denoting the resulting action of this open transition. Pred and Post are defined similarly to Definition 2.4. We use $\mathcal{WT}$ to range over sets of weak open transitions.*

A weak open automaton $\langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT}\rangle\!\rangle$ is similar to an open automaton except that $\mathcal{WT}$ is a set of weak open transitions over J and $\mathcal{S}$.

Definition 2.8 (Building a weak open automaton) *Let $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T}\rangle\!\rangle$ be an open automaton. The weak open automaton* derived *from A is an open automaton $\langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT}\rangle\!\rangle$, where $\mathcal{WT}$ is derived from $\mathcal{T}$ using the 3 semantic rules as follows:*

$$\frac{}{\dfrac{\emptyset, True, ()}{s \overset{\tau}{\Rightarrow} s}} \in \mathcal{WT} \qquad \textbf{WT1}$$

$$\frac{\dfrac{\overline{\beta}, Pred, Post}{s \overset{\alpha}{\longrightarrow} s'} \in \mathcal{T}}{\dfrac{(\overline{\beta})^{\nabla}, Pred, Post}{s \overset{\alpha}{\Rightarrow} s'} \in \mathcal{WT}} \qquad \textbf{WT2}$$

$$\frac{\dfrac{\overline{\gamma_1}, Pred_1, Post_1}{s \overset{\tau}{\Rightarrow} s_1} \in \mathcal{WT} \quad \dfrac{\overline{\gamma_2}, Pred_2, Post_2}{s_1 \overset{\alpha}{\Rightarrow} s_2} \in \mathcal{WT} \quad \dfrac{\overline{\gamma_3}, Pred_3, Post_3}{s_2 \overset{\tau}{\Rightarrow} s'} \in \mathcal{WT}}{\begin{array}{c} Pred = Pred_1 \wedge Pred_2\{\!\{Post_1\}\!\} \wedge Pred_3\{\!\{Post_2 \otimes Post_1\}\!\} \\ \overline{\gamma} = \overline{\gamma_1} \cup \overline{\gamma_2}\{\!\{Post_1\}\!\} \cup \overline{\gamma_3}\{\!\{Post_2 \otimes Post_1\}\!\} \qquad \alpha' = \alpha\{\!\{Post_1\}\!\} \\ \hline \dfrac{\overline{\gamma}, Pred, Post_3 \otimes Post_2 \otimes Post_1}{s \overset{\alpha'}{\Rightarrow} s'} \in \mathcal{WT} \end{array}} \qquad \textbf{WT3}$$

2.4 Running Example

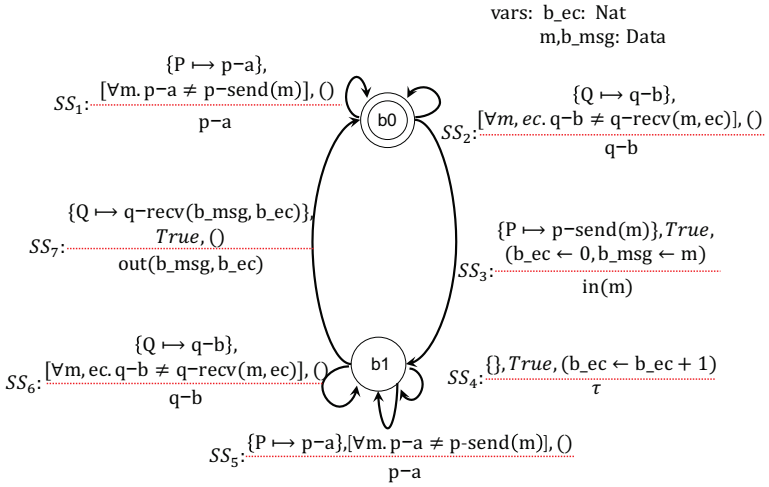


Figure 2: Open Automaton of the protocol Specification

The OA in Figure 2, which is computed from pNet in Figure 1, describes the behaviour of the Specification of the communication protocol. Consider e.g. the Open Transition SS_3 from state b0 to b1, the Hole P is involved and performs action `p-send(m)`, its predicate is True, and its Post assigns variables `b_ec` and `b_msg`, while the resulting action visible here is `in(m)`.

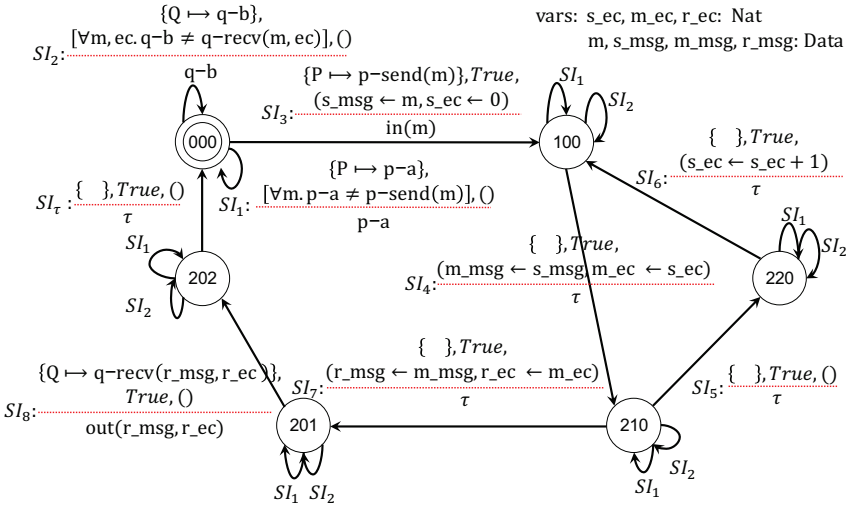


Figure 3: Open Automaton of the Implementation

The OA in Figure 3 represents the behaviour of the Implementation. In the Implementation, the system becomes more detailed and complex. The states 100, 220, 210, and their actions reflect the internal structure of the protocol. The message data and error counter pass through several variables. We do not need to pay too much attention to its mechanical details, apart

from knowing it is just one of all possible implementations.

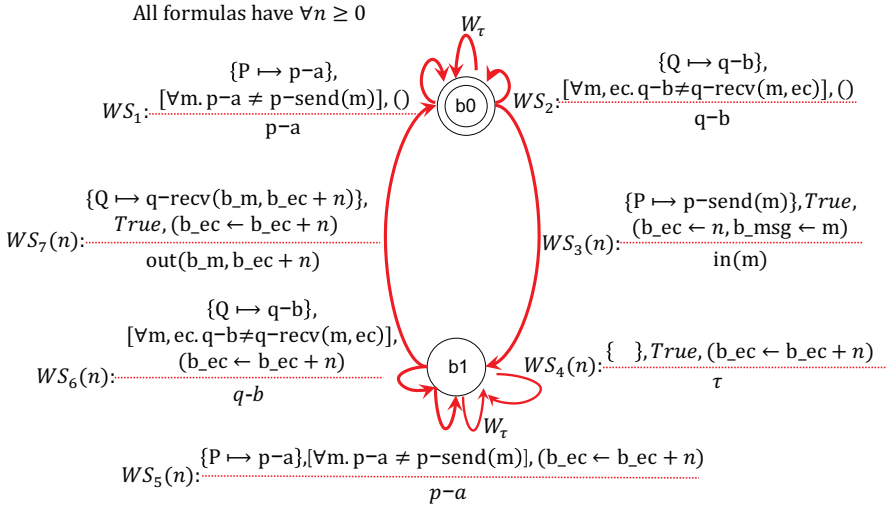


Figure 4: Weak Open Automaton of the Specification

2.4.1 Implementation WOA

According to Definition 2.8, Weak OAs in Figure 4 is built from (strong) OAs from Figure 2; Weak OAs in Figure 5 is built from (strong) OAs from Figure 3.

According to rule **WT1**, each state has a Self-Loop *tau* transition.

$$W_\tau = \frac{\{\}, True, ()}{s \overset{\tau}{\Rightarrow} s}, \forall s \in \mathcal{S}$$

Then each OT can be transformed to a WOT by rule **WT2**. For example, we get WS_1, in Figure 4, built from SS_1, in Figure 2.

$$WS_1 = \frac{\{\texttt{P}\mapsto\texttt{p-a}\}, [\forall \texttt{m.p-a} \neq \texttt{p-send(m)}], ()}{b0 \overset{\texttt{p-a}}{\Longrightarrow} b0}$$

It's the same for WS_2, WS_3, WS_5, WS_6, and all the red arrow lines in Figure 5.

Rule **WT3** is the most interesting one. Let us take WS_4 as an example. We have SS_4 in Figure 2. This OT has the same source state and target state b1 with τ action. Nevertheless, this OT can occur infinitely, which forms a transition sequence like SS_4^*.

We can build a WOT, $WS_4(2)$ from the transition sequence $SS_4; SS_4$ by rule **WT3**:

$$WS_4(2) = \frac{\{\}, True, (\texttt{b\_ec} \leftarrow \texttt{b\_ec + 2})}{b1 \overset{\tau}{\Rightarrow} b1}$$

Then we use mathematical induction to get $WS_4(n)$ from SS_4^*. Here n is a natural number indicating how many times it goes through SS_4. Actually, $WS_4(n)$ is a meta weak open transition

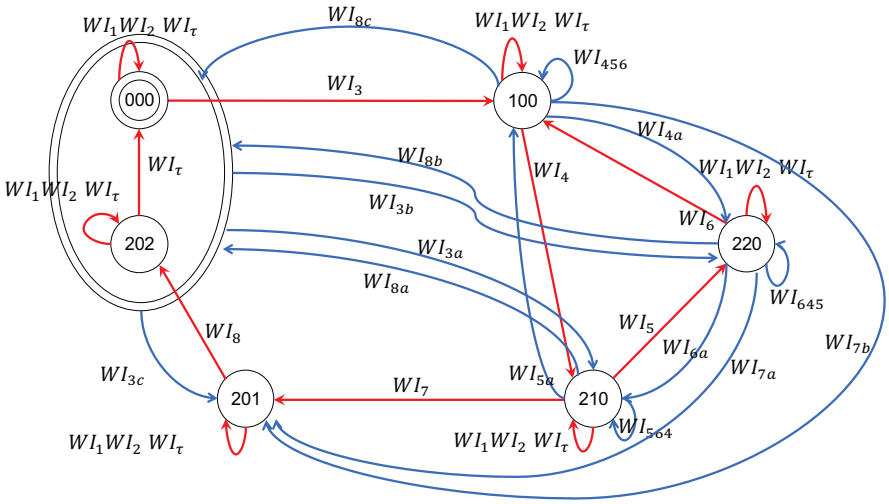


Figure 5: Weak Open Automaton of the Implementation

(meta-WOT), and variable n is a meta-Variable, which will be formalized in the next section (same as $WS_3(n)$, $WS_5(n)$, $WS_6(n)$, $WS_7(n)$).

In Figure 5 , there is a circle covering states 000 and 202, which means these two states are equivalent in some sense. This is because the only transition connecting 000 and 202 is WI_τ, and they all have *similar* self-loop and outgoing transitions. In other words, the environment (observer or other components at the same or higher level of pNet) can not distinguish which state the current system visits.

Let us show some details of the weak transitions in Figure 5. Most of them are also meta-WOTs. The full set of weak transitions is listed in Appendix A.

$WI_{456}(n)$ is constructed from the transition sequence $(WI_4WT_5\,WI_6)^*$.

$$WI_{456}(n) = \frac{\{\}, True, (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \stackrel{\tau}{\Rightarrow} 100}, \forall n \geqslant 0$$

WI_4 is transformed from SI_4.

$$WI_4 = \frac{\{\}, True, (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec}, \texttt{s\_ec} \leftarrow \texttt{s\_ec})}{100 \stackrel{\tau}{\Rightarrow} 210}$$

$WI_4(n)$ is constructed by transition sequence $WI_{456}(n)$; WI_4. It is easy to see that WI_4 is a particular part of $WI_4(n)$ when $n = 0$. So, it is enough only to keep $WI_4(n)$ and omit WI_4.

$$WI_4(n) = \frac{\{\}, True, (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec} + n, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \stackrel{\tau}{\Rightarrow} 210}, \forall n \geqslant 0$$

$WI_3(n)$ is constructed by transition sequence WI_τ; WI_3; $WI_{456}(n)$:

$$WI_3(n) = \frac{\{\texttt{P}\mapsto\texttt{p-send(m)}\}, True, (\texttt{s\_msg} \leftarrow \texttt{m}, \texttt{s\_ec} \leftarrow n)}{\{000, 202\} \stackrel{\texttt{in(m)}}{\Longrightarrow} 100}, \forall n \geqslant 0$$

$WI_{3a}(n)$ is constructed by transition sequence WI_τ; WI_3; $WI_{456}(n)$; WI_4:

$$WI_{3a}(n) = \frac{\{\texttt{P}\mapsto\texttt{p-send(m)}\}, True, (\texttt{m\_msg} \leftarrow \texttt{m}, \texttt{m\_ec} \leftarrow n, \texttt{s\_ec} \leftarrow n)}{\{000, 202\} \xRightarrow{\texttt{in(m)}} 210}, \forall n \geqslant 0$$

For all τ transitions above, we have a similar WOT that include a non-τ move from an external action of P or Q, for example:

$$WI_{4P}(n) =$$

$$\frac{\{\texttt{P}\mapsto\texttt{p-a}\}, [\forall \texttt{m}.\texttt{p-a} \neq \texttt{p-send(m)}], (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec} + n, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \xRightarrow{\texttt{p-a}} 210}$$

Similarly, we can build all other transitions.

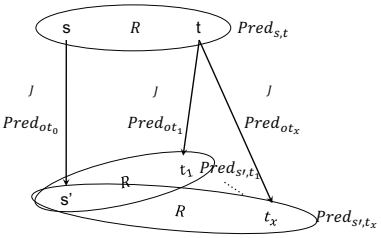
2.5 Bisimulation of Open Automata

In this section, we introduce two kinds of bisimulation relations for the Open Automata: Strong FH-Bisimulation [9, 10] and Weak FH-Bisimulation [1]. Each comes in two flavours: an extensional version, infinite by nature, that is better suited for mathematical proofs, and an intentional version, based on finite encodings of the OAs, that is better fit for algorithms. All of these relations are defined as sets of Triples $(s, t|Pred_{s,t})$, between states of two OAs, and a predicate relating the values of the OAs variables to make these states equivalent.

From these former definitions, our contributions in the next sections will target a new intentional encoding for Weak open transitions, more powerful, and various approaches to check Weak bisimulation for these OAs.

Definition 2.9 (Strong FH-Bisimulation)

Suppose $A_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1 \rangle\!\rangle$ *and* $A_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2 \rangle\!\rangle$ *are open automata with identical Holes of the same sort, with disjoint sets of variables.*
Then $\mathcal{R}$ *is an FH-Bisimulation if and only if for any states* $s \in \mathcal{S}_1$ *and* $t \in \mathcal{S}_2$, $(s, t|Pred_{s,t}) \in \mathcal{R}$, *as figure right side,*



we have the following:

- *For any open transition* ot *in* $\mathcal{T}_1$*:* $\dfrac{\beta_j^{j\in J'}, Pred_{ot}, Post_{ot}}{s \xrightarrow{\alpha} s'}$

  there exist open transitions $ot_x^{x\in X} \subseteq \mathcal{T}_2$*:* $\dfrac{\beta_{jx}^{j\in J_x}, Pred_{ot_x}, Post_{ot_x}}{t \xrightarrow{\alpha_x} t'_x}$

  such that $\forall x, J' = J_x$, *and there exists* $Pred_{s',t'_x}$
  such that $(s', t_x|Pred_{s',t'_x}) \in \mathcal{R}$; *and*
  $Pred_{s,t} \wedge Pred_{ot} \implies \bigvee_{x\in X} \left(\forall j.\beta_j = \beta_{jx} \wedge Pred_{ot_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t'_x} \{\!\{Post_{ot} \uplus Post_{ot_x}\}\!\}\right)$

- *and symmetrically any open transition from* t *in* $\mathcal{T}_2$ *can be covered by a set of transitions from* s *in* $\mathcal{T}_1$.

Two open automata (A_1 and A_2 mentioned above) are related by FH-Bisimulation w.r.t. an initial condition $Pred_{s_0,t_0}$ if there exists an FH-Bisimulation $\mathcal{R}$ between them, and their initial states are in $\mathcal{R}$ with a predicate $Pred_{s_0,t_0}$.

When the OA has some variables with initial values, the initial predicate, by default, can be derived from the initial values:

$$Pred_{s_0,t_0} = \bigvee_{v \in V} \Big(if\ defined(init(v)) \quad then\ v = init(v)\ else\ true \Big)$$

But one could also want to specify a stronger initial predicate, and define parameterised equivalences.

The following properties were introduced and proven in [9]:

Theorem 2.10 (FH-Bisimulation is an equivalence) *Suppose $\mathcal{R}$ is an FH-Bisimualtion. Then $\mathcal{R}$ is an equivalence, that is, $\mathcal{R}$ is reflexive, symmetric and transitive.*

Compositionality: In the same work, it was also shown that the FH-Bisimulation has important compositionality properties, namely that it is preserved by instantiation of the Holes of the system under analysis. So properties of subsystems (modules, components, processes) can be proven locally, by bisimulation checking or model-checking, and can be used as such after composition of the whole system. This was shown in [9] in the case of pNet systems, but it could also be applied to many other formalisms, e.g. in process algebras, for composition using (value-passing) CCS operators.

2.5.1 StrFH-Bisimulation

StrFH-Bisimulation, introduced by [10], means Structural Formal Hypothesis Bisimulation, which is based on Strong FH-Bisimulation from [9]. It is a kind of structural strong bisimulation equivalence between finitely defined open automata that are not necessarily closed under substitution.

The formal definition of StrFH-Bisimulation is almost the same as (Strong) FH-Bisimulation, but the proof obligation part includes additional quantifiers:

$$\forall otvars(ot).\Bigg\{ Pred_{s,t} \wedge Pred_{ot} \implies \bigvee_{x \in X} \Big[\exists otvars(ot_x).$$

$$\Big(\forall j.\beta_j = \beta_{jx} \wedge Pred_{ot_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t'_x} \{\!\{ Post_{ot} \uplus Post_{ot_x} \}\!\} \Big) \Big] \Bigg\}$$

This formula requires some explanations. We must distinguish here two kinds of variables: The 'state variables' are parameters to the automaton; intuitively, they allow us to encode a set of states (potentially infinite) into a single state, by using some data variables. These are typically the variables that will appear in the Triples predicates: parameterized states are related only for some specific values of the automaton variables. But we also have some other variables appearing locally in the open transitions themselves, that we call local variables and denote *otvars* here (the 'input variables' from section 2.1.1). Here an open transition using local variables also encode an infinite set of ground transitions. They play a specific role in the bisimulation proof obligation: for any instantiation of the local variables of *ot* above, there should exist some values of the local vars of the ot_x so that the formula holds. This is the meaning of the quantifiers in the proof obligation.

Let ot be an open transition of an open automaton A. Recall from Def. 2.4 that $vars(ot)$ denotes the variables (from A) occurring in ot, and $otvars(ot)$ the set of local variables in ot. Now the formula $\forall otvars(ot).\phi$ (resp. $\exists otvars(ot).\phi$) means that for any valuation of all the variables in $otvars(ot)$, the inside formula is true. For example, for an open transition ot which contains local variables $otvars(ot) = \{x, y, z\}$, then $\forall otvars(ot)$ is a shortcut for $\forall x, y, z.\phi$.

2.5.2 Weak FH-Bisimulation

This is an equivalence relation similar to the strong FH-Bisimulation between two open automata, relating the open transition of open automata with transitions of the weak open automaton derived from the other.

Definition 2.11 (Weak FH-Bisimulation) *Let $A_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1 \rangle\!\rangle$ and $A_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2 \rangle\!\rangle$ be open automaton with disjoint sets of variables. Let $woa_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{WT}_1 \rangle\!\rangle$ and $woa_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{WT}_2 \rangle\!\rangle$ be the weak open automata derived from A_1 and A_2 respectively. Let $\mathcal{R}$ a relation over $\mathcal{S}_1$ and $\mathcal{S}_2$, as in Definition 2.9.*

Then $\mathcal{R}$ is a Weak FH-Bisimulation iff for any states $s \in \mathcal{S}_1$ and $t \in \mathcal{S}_2$ such that $(s, t|Pred_{s,t}) \in \mathcal{R}$, we have the following:

- *For any open transition ot in $\mathcal{T}_1$: $\dfrac{\beta_j^{j\in J'}, Pred_{ot}, Post_{ot}}{s \xrightarrow{\alpha} s'}$*

  there exist weak open transitions $wot_x^{x\in X} \subseteq \mathcal{WT}_2$: $\dfrac{\gamma_{jx}^{j\in J_x}, Pred_{ot_x}, Post_{ot_x}}{t \xRightarrow{\alpha_x} t'_x}$

  such that $\forall x, \{j \in J' | \beta_j \neq \tau\} = J_x$,
  $(s', t'_x | Pred_{s',t'_x}) \in \mathcal{R}$; *and*
  $$Pred_{s,t} \wedge Pred_{ot} \implies \bigvee_{x\in X} \left(\forall j \in J_x.(\beta_j)^{\nabla} = \gamma_{jx} \wedge Pred_{ot_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t'_x} \{\!\{ Post_{ot} \uplus Post_{ot_x} \}\!\} \right)$$

- *and symmetrically any open transition from t in $\mathcal{T}_2$ can be covered by a set of weak transitions from s in $\mathcal{WT}_1$.*

3 Two approaches for Checking Weak FH-Bisimulation

This section will introduce some useful definitions that will be widely used in constructing WOT and WOA in practical needs, especially the *Weak StrFH-Bisimulation*. Furthermore, in the final sub-section, we discuss the two searching styles to check weak bisimulation, building a complete WOA before checking or searching some expected WOTs on-the-fly on demand. Our conclusion will be that it is easier and more efficient to implement the second way of checking weak bisimulation.

3.1 Preliminaries

In this subsection, we will introduce Weak StrFH-Bisimulation first and then other definitions. For short, we use WOT standing for weak open transition, WOA for weak open automaton, OT for open transition, and OA for open automaton. $\mathcal{T}$ is a set of OTs, and $\mathcal{WT}$ is a set of WOTs.

3.1.1 Weak StrFH-Bisimulation

Definition 2.11 suits mathematical reasoning but is not useful in practice, because of it deals with infinite sets. So, we define Weak StrFH-Bisimulation from Weak FH-Bisimulation, just like [10] defining the StrFH-Bisimulation from FH-Bisimulation.

Definition 3.1 (Weak StrFH-Bisimulation) *Let $A_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1 \rangle\!\rangle$ and $A_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2 \rangle\!\rangle$ be open automata with disjoint sets of variables. Let $\langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{WT}_1 \rangle\!\rangle$ and $\langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{WT}_2 \rangle\!\rangle$ be the weak open automata derived from A_1 and A_2 respectively. Let $\mathcal{R}$ a relation over $\mathcal{S}_1$ and $\mathcal{S}_2$, as in Definition 2.9.*

Then $\mathcal{R}$ is a Weak StrFH-Bisimulation iff for any states $s \in \mathcal{S}_1$ and $t \in \mathcal{S}_2$ such that $(s, t|Pred_{s,t}) \in \mathcal{R}$, we have the following:

- *For any open transition ot in $\mathcal{T}_1$: $\dfrac{\beta_j^{j\in J'}, Pred_{ot}, Post_{ot}}{s \xrightarrow{\alpha} s'}$*

  there exist weak open transitions $wot_x^{x\in X} \subseteq \mathcal{WT}_2$: $\dfrac{\gamma_{jx}^{j\in J_x}, Pred_{ot_x}, Post_{ot_x}}{t \xRightarrow{\alpha_x} t'_x}$

  such that $\forall x, \{j \in J' | \beta_j \neq \tau\} = J_x$,
  $(s', t'_x | Pred_{s',t'_x}) \in \mathcal{R}$; and

$$\forall\ otvars(ot).\Big\{ Pred_{s,t} \wedge Pred_{ot} \implies \bigvee_{x\in X} \Big[\exists\ otvars(ot_x).$$

$$\Big(\forall j \in J_x.(\beta_j)^{\nabla} = \gamma_{jx} \wedge Pred_{ot_x} \wedge\ \alpha = \alpha_x \wedge Pred_{s',t'_x} \{\!\!\{ Post_{ot} \uplus Post_{ot_x} \}\!\!\}\Big)\Big]\Big\}$$

- *and symmetrically any open transition from t in $\mathcal{T}_2$ can be covered by a set of weak transitions from s in $\mathcal{WT}_1$.*

Two OA are Weak StrFH-Bisimilar if there exists a Weak StrFH-Bisimulation between their associated automata and their initial states are in the relation, i.e., the predicate associated with the relation between the initial states is True.

3.1.2 Meta-Variables

There is an interesting kind of transitions, namely Self-Loop Transitions. They can be OT or WOT, defined formally as follow:

Definition 3.2 (Self-Loop Transition) *A Self-Loop transition is a (weak) open transition with the same source state and target state.*

Formally, as follows:

$$Self\text{-}Loop\ OT = \frac{\overline{\beta}, Pred, Post}{s \xrightarrow{\alpha} s} \qquad Self\text{-}Loop\ WOT = \frac{\overline{\gamma}, Pred, Post}{s \xRightarrow{\alpha} s}$$

All the $\mathcal{WT}_\tau$ built from **WT1** are Self-Loop transitions. According to **WT3**, a weak open transition can be built from a sequence of WOTs in which the previous one's target state is the

next one's source state. If the first WOT's source state is the same as the last WOT's target state in the sequence, then the sequence of WOTs has a loop from which we can construct a Self-Loop WOT by **WT3**. This kind of transition sequence is called a 'loop sequence'.

If we find Self-Loop weak open transitions when applying WOT construction rules, Definition 2.8, then the space of $\mathcal{WT}$ is theoretically infinite. Because the Self-Loop WOT can append on itself by **WT3** infinitely many times, it can produce infinitely many WOTs. Then, we can use meta-Variables to encode the set of WOTs.

For example, a segment of the Post in $WI_{456}(n)$ in Figure 5, is ($\texttt{s\_ec} \leftarrow \texttt{s\_ec} + n$). Here we use a meta-Variable n to represent the infinite possible values of s_ec.

So we now extend the definitions of weak transitions and weak automata to include meta-Variables. Each meta weak transition has its own local set of meta-Variables. We add a set of meta-Variables in open automaton structure, to ensure that they do not conflict with "normal" automata variables.

Definition 3.3 (Meta-WOA and meta-WOT) *A Meta-WOA, the abbreviation of meta weak open automaton, is a structure $A = \langle\!\langle J, \mathcal{S}, s_0, V, M, \mathcal{MWT} \rangle\!\rangle$ where:*

- *J, $\mathcal{S}$, s_0 is similar to an open automaton.*
- *V is the set of variables, M is the set of meta-Variables, with $V \cup M = \emptyset$. We denote $V = vars(A)$, $M = mvars(A)$.*
- *$\mathcal{MWT}$ is a set of meta weak open transitions which is similar to the weak open transition but needs a set of meta-Variables to express its semantic meaning.*

A meta weak open transition, (meta-WOT or mWOT for short), is a structure similar to weak open transitions, of the form:

$$mwot(M') = \frac{\gamma_j^{j\in J'}, Pred, Post}{s \overset{\alpha}{\Rightarrow} s'}$$

This is similar to Definition 2.7, but includes a set of meta-Variables $M' \subseteq M$ that can be used in all the expressions γ, Pred, Post, and α. Each $mv \in M$ has its domain, $dom(mv)$. Notice that when the set of meta-Variables is $\emptyset$, or each meta-Variable has one single legal valuation, then the meta-WOT is a normal WOT.

So now we have three types of variables that can occur in a meta-WOT: OA's variables from V and meta-Variables from M', and local variables of the OT, $otvars(mwot(M'))$. Let us call otmvars the set of variables local to one meta-WOT:
$otmvars(mwot(M')) = M' \cup otvars(mwot(M'))$.

One way to construct Meta-WOTs is to use repeatedly the construction rules of Definition 2.8. When a transition sequence has infinite or finite repetitive segments and applies **WT3**, caused by Self-Loop transitions, we can infer meta-Variables from the result WOT and build a meta-WOT.
Remark that this "construction" may be difficult to encode as an algorithm in the general case, when a set of WOTs are generated along such an infinite path. If the operations in the transition's assignments are using non-trivial data, then maybe Abstract Interpretation and Generalization approaches could be used to find corresponding meta-variables. But this will never be a complete and decidable construction.

When each meta-Variable of a meta-WOT is assigned a particular value, then it becomes a normal weak open transition. So, normal WOT is a special kind of meta-WOT with $M = \emptyset$

or $\forall mv \in M.mv \leftarrow value_m$. In other words, before any assignment, a meta-WOT represents a set of WOTs. If $\exists mv \in M$ such that $dom(mv)$ is infinite, then the set of WOTs is infinite. Otherwise, a meta-WOT represents a finite set of WOTs.

As we have shown in the previous section, $WI_{456}(n)$ in Figure 5, is a meta-WOT with meta-Variables $\{n\}$, a natural number.

$$WI_{456*}(n) = \frac{\{\}, True, (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \overset{\tau}{\Rightarrow} 100}, \forall n \geqslant 0$$

In most cases, the meta-Variables are generated by Self-Loop WOTs and the meta-Variables relate to the number of loop times. $WI_{456}(n)$ is a Self-Loop WOT with a meta-Variable built by the WOT sequence $WI_4\, WI_5\, WI_6$ for n times.

Property 3.4 (Space of WOT and meta-WOT) *Given an automaton $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$, woa is derived from A, where $woa = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT} \rangle\!\rangle$, and mwoa is a meta-WOA corresponding to woa, where $mwoa = \langle\!\langle J, \mathcal{S}, s_0, V, M, \mathcal{MWT} \rangle\!\rangle$. Then:*

- $\mathcal{WT} = \{mWOT_x(M_x)\{\!\{\sigma\}\!\} \mid mWOT_x(M_x) \in \mathcal{MWT}, \sigma \in \Sigma\}$ *where* $\bigcup_{x \in X} M_x = M$, *and X is the index set of* $\mathcal{MWT}$.
- $\forall\ wot \in \mathcal{WT}, \exists\ mwot(M_x) \in \mathcal{MWT}, \exists\ \sigma \in \Sigma.\ wot = mwot(M_x)\{\!\{\sigma\}\!\}$, *where σ is a legal valuation for each meta-Variables in M_x.*

As illustrated in Figure 6, a meta-WOT represents a set of WOTs, but does not change the space of $\mathcal{WT}$. Here, *mwoa* denotes a meta-WOA and *mwot* a meta-WOT.

The first item in Property 3.4 means that if you replace each meta-WOT in $\mathcal{MWT}$ using the set (possibly infinite) of all possible values of its meta-Variables, then you get exactly the set of WOTs in $\mathcal{WT}$.

The second item means that for any *wot*, there at least exists one *mwot* with assignments σ such that $mwot\{\!\{\sigma\}\!\} = wot$.

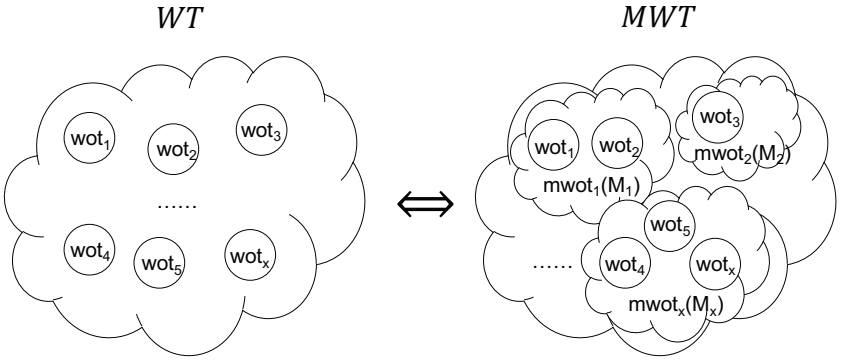


Figure 6: $\mathcal{WT}$ and $\mathcal{MWT}$

3.2 meta FH-Bisimulation

In this section, we provide a new relation between meta-WOAs, defined formally as:

Definition 3.5 (meta FH-Bisimulation) *Let* $mwoa_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, M_1, \mathcal{MWT}_1 \rangle\!\rangle$ *and* $mwoa_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, M_2, \mathcal{MWT}_2 \rangle\!\rangle$ *be two meta weak automata.*

A TripleSet $\mathcal{R}$ *is a meta FH-Bisimulation iff for any states* $s \in \mathcal{S}_1$ *and* $t \in \mathcal{S}_2$, $(s, t | Pred_{s,t}) \in \mathcal{R}$, *we have the following:*

- *For any meta weak open transition*

  $mwot(M_1) \in \mathcal{MWT}_1$: $\dfrac{\gamma_j^{j\in J'}, Pred_{mwot}, Post_{mwot}}{s \overset{\alpha}{\Rightarrow} s'}$

  there exist meta weak open transitions

  $mwot_x^{x\in X}(M_2) \subseteq \mathcal{MWT}_2$: $\dfrac{\gamma_{jx}^{j\in J_x}, Pred_{mwot_x}, Post_{mwot_x}}{t \overset{\alpha_x}{\Rightarrow} t'_x}$

  such that $\forall x, J' = J_x, Pred_{s',t_x}.(s', t'_x | Pred_{s',t'_x}) \in \mathcal{R}$; *and*

$$\forall\ otmvars(mwot).\Big\{ Pred_{s,t} \wedge Pred_{mwot} \implies \bigvee_{x\in X} \Big[\exists\ otmvars(mwot_x).$$

$$\Big(\forall j.\gamma_j = \gamma_{jx} \ \wedge Pred_{mwot_x} \wedge \alpha{=}\alpha_x \wedge Pred_{s',t'_x}\{\!\{Post_{mwot} \uplus Post_{mwot_x}\}\!\} \Big) \Big] \Big\}$$

- *and symmetrically any weak open transition from* t *in* $\mathcal{MWT}_2$ *can be covered by a set of transitions from* s *in* $\mathcal{MWT}_1$.

Two OAs are meta FH-Bisimilar w.r.t. some initial condition $Pred_0$ *if there exists a meta FH-Bisimulation relation between their associated automata, and their initial states are in the relation, i.e., the predicate associated with the relation between the initial states is* $Pred_0$.

Remark that the Triple predicates (and here $Pred_0$*) can only include standard automata variables, not meta-Variables.*

3.3 Expansion

Here, we define an Expansion function $\mathcal{E}$, such that for any meta-weak open automaton, we can use this function to get a corresponding semantic (weak) open automaton.

Definition 3.6 (Expansion) *For any meta-Weak open automaton* $mwoa = \langle\!\langle J, \mathcal{S}, s_0, V, M, \mathcal{MWT} \rangle\!\rangle$ *we define an Expansion Function* $\mathcal{E}$ *that builds a Weak Open Automaton* $woa = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT} \rangle\!\rangle$*:* $\mathcal{E} : \mathcal{MWT}(M) \to PowerSet(\mathcal{WT})$

The only differences between mwoa and woa above are on the transition set and the meta-Variables. For any $mwot(M_1) \in \mathcal{MWT}, M_1 \subseteq M$, *we have :*

$$mwot(M_1) = \frac{\gamma_j^{j\in J}, Pred_{mwot}, Post_{mwot}}{s \overset{\alpha}{\Rightarrow} s'}$$

Let Σ *be the set of all legal substitutions for variables in* M_1, *where* $\forall \sigma = (mv_k \leftarrow x)^{k\in K} \in \Sigma$, $\forall k \in K$, $mv_k \in M_1$ *and* $x \in dom(mv_k)$. *Apply* σ *on mwot, we get* $mwot(M_1)\{\!\{\sigma\}\!\}$*:*

$$\frac{\gamma_j^{j\in J}\{\!\{\sigma\}\!\}, Pred_{mwot}\{\!\{\sigma\}\!\}, Post_{mwot}\{\!\{\sigma\}\!\}}{s \overset{\alpha\{\sigma\}}{\Longrightarrow} s'} \in \mathcal{WT}$$

Then we define:

$$\mathcal{E}(mwot(M_1)) = \{mwot(M_1)\{\!\{\sigma\}\!\} \mid \sigma \in \Sigma\} \subseteq PowerSet(\mathcal{WT})$$

$$\mathcal{E}(\langle\!\langle J, S, s_0, V, M, \mathcal{MWT}\rangle\!\rangle) = \langle\!\langle J, S, s_0, V, \{\mathcal{E}(mwot) | mwot \in \mathcal{MWT}\}\rangle\!\rangle$$

So $\mathcal{E}$ expands a meta weak OT into a potentially infinite set of weak OTs; and a meta-WOA into a WOA with the same holes, states, initial state, 'non-meta' variables, and a potentially infinite set of WOTs. Now we prove that $\mathcal{E}$ preserves Weak Bisimulation:

Theorem 3.7 *Let oa_1, oa_2 be 2 open automata, and woa_1, woa_2 their weak version, constructed with Definition 2.8. Let $mwoa_1, mwoa_2$ be 2 meta open automata such that $\mathcal{E}(mwoa_1) = woa_1 \wedge \mathcal{E}(mwoa_2) = woa_2$. Then the meta FH-Bisimulation relation on $mwoa_1$ and $mwoa_2$ is equivalent to the Weak FH-Bisimulation relation on oa_1 and oa_2, formally the two following statements are equivalent:*

- *there exists a Tripleset $w\mathcal{R}$ that is a Weak FH-Bisimulation between oa_1 and oa_2, under some initial predicate $Pred_0$,*
- *there exists a Tripleset $mw\mathcal{R}$ that is a meta FH-Bisimulation between $mwoa_1$ and $mwoa_2$, with the same initial predicate $Pred_0$.*

Property 3.8 *As a corollary, the expansion of meta OAs preserves weak bisimulation, so checking meta FH-Bisimulation can be used to check Weak FH-Bisimulation on a finitary representation of the weak OAs.*

In [1] (Appendix B, Lemma 5), we have already proven that Weak FH-Bisimulation is equivalent to the following alternative definition of weak bisimulation. We will use this result as a basis to the proof of Theorem 3.7:

Definition 3.9 (Alternate Weak FH-Bisimulation) *Let $oa_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1\rangle\!\rangle$ and $oa_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2\rangle\!\rangle$ be open automata with the same set of Holes J, and with disjoint sets of variables; $woa_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{WT}_1\rangle\!\rangle$ and $woa_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{WT}_2\rangle\!\rangle$ be the weak open automata derived from oa_1 and oa_2 respectively. Then a Tripleset $aw\mathcal{R}$ is a Weak FH-Bisimulation, iff for any pair of states $s \in \mathcal{S}_1$ and $t \in \mathcal{S}_2$ such that $(s, t | Pred_{s,t}) \in aw\mathcal{R}$, we have:*

- *For any open transition wot in $\mathcal{WT}_1$: $\dfrac{\gamma_j^{j\in J'}, Pred_{wot}, Post_{wot}}{s \overset{\alpha}{\Rightarrow} s'}$*

  there exist weak open transitions $wot_x^{x\in X} \subseteq \mathcal{WT}_2$: $\dfrac{\gamma_{jx}^{j\in J_x}, Pred_{wot_x}, Post_{wot_x}}{t \overset{\alpha_x}{\Rightarrow} t'_x}$

  such that $\forall x, J' = J_x, (s', t'_x | Pred_{s',t'_x}) \in aw\mathcal{R}$; and

  $$Pred_{s,t} \wedge Pred_{wot} \implies \bigvee_{x\in X} \left(\forall j \in J_x . \gamma_j = \gamma_{jx} \wedge Pred_{wot_x} \wedge \alpha = \alpha_x \wedge Pred_{s',t'_x} \{\!\{Post_{wot} \uplus Post_{wot_x}\}\!\} \right)$$

- *and symmetrically any open transition from wot in $\mathcal{WT}_2$ can be covered by a set of weak transitions from s in $\mathcal{WT}_1$.*

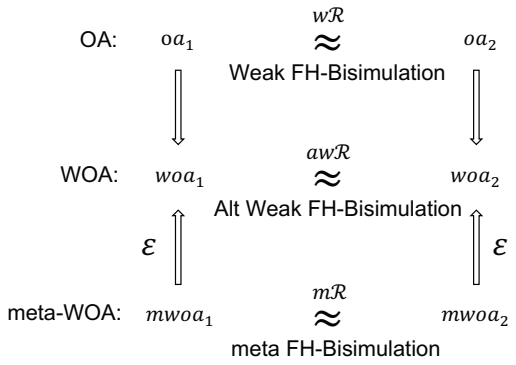


Figure 7: Schema of Theorem 3.7 proof

Proof 3.10 *of Theorem 3.7:*

The definition of Weak FH-Bisimulation in Definition 2.11 is equivalent to the meta FH-Bisimulation in Definition 3.5.

We use Definition 3.9 to simplify the proof of Theorem 3.7, with the schema from Figure 7.

Let $oa_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{T}_1 \rangle\!\rangle$, $oa_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{T}_2 \rangle\!\rangle$, $woa_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, \mathcal{WT}_1 \rangle\!\rangle$, $woa_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, \mathcal{WT}_2 \rangle\!\rangle$, $mwoa_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, M_1, \mathcal{MWT}_1 \rangle\!\rangle$, $mwoa_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, M_2, \mathcal{MWT}_2 \rangle\!\rangle$ *be open automata verifying the hypotheses of Theorem 3.7.*

Suppose $mwoa_1$ *and* $mwoa_2$ *have meta FH-Bisimulation relation (Def. 3.5) with relation Triples* $m\mathcal{R}$.

We want to prove that woa_1 *and* woa_2 *are related by some relation* $aw\mathcal{R}$, *and that their initial states are equivalent:* $(s_0, t_0 | Pred_{s_0,t_0}) \in aw\mathcal{R}$. *As there are no meta-Variables in the Tripleset predicates, we keep* $aw\mathcal{R} = m\mathcal{R}$.

Let wot_1 *be any weak OT of* woa_1. *By hypothesis, it is obtained as the expansion of some meta-WOT* $mwot_1(M_1')$ *of* woa_1*:*

$$mwot_1(M_1') = \frac{\gamma_j^{j\in J}, Pred_1, Post_1}{s \overset{\alpha_1}{\Longrightarrow} s'}, M_1' \subseteq M_1$$

and

$$\begin{aligned} wot_1 &= mwot(M_1')\{\!\{\sigma_1\}\!\} \\ &= \frac{\gamma_j^{j\in J}\{\!\{\sigma_1\}\!\}, Pred_1\{\!\{\sigma_1\}\!\}, Post_1\{\!\{\sigma_1\}\!\}}{s \overset{\alpha_1\{\sigma_1\}}{\Longrightarrow} s'} \in \mathcal{WT}_1 \end{aligned}$$

for some substitution σ_1 *of the variables in* M_1'.

By meta FH-Bisimulation, we know that there exists a set $MWT2_X = \{mwot_{2x}(M_{2x}')\}^{x\in X}$ *of meta-WOTs from* $mwoa_2$, *with:*

$$mwot_{2x}(M_{2x}') = \frac{\gamma_{j'}^{j'\in J}, Pred_{2x}, Post_{2x}}{t \overset{\alpha_{2x}}{\Longrightarrow} t_x'},$$
$$where\ mwot_{2x} \in \mathcal{MWT}_2, M_{2x}' \subseteq M_2$$

And its extension is :

$$WT2_Y = \{mwot_{2x}(M'_{2x})\{\!\{\sigma_2\}\!\}\}^{\sigma_2\in\Sigma_2, x\in X}$$
$$wot_{2x\ \sigma_2} = mwot_{2x}(M'_{2x})\{\!\{\sigma_2\}\!\}$$
$$= \frac{\gamma_{jx}^{jx\in J}\{\!\{\sigma_2\}\!\}, Pred_{2x}\{\!\{\sigma_2\}\!\}, Post_{2x}\{\!\{\sigma_2\}\!\}}{t \xrightarrow{\alpha_{2x}\ \{\sigma_2\}} t'_x}$$

Naturally, this set is very large, it contains all possible instantiations of the meta-WOTs covering wot_1. *But among these, it contains those instantiations that we are interested in, so let's keep all of them. Consider the set $WT2_Y$ as defined above. It is an extension of $MWT2_X$, so it is a subset of $\mathcal{WT}_2$, the weak OTs of woa_2. We will prove that it* covers wot_1.

By hypothesis, we know that:

- *all meta-WOTs in $MWT2_X$ have the same set of working holes than $mwot_1$, and this is preserved by expansion, so all WOTs in $WT2_Y$ have the same set of working holes than wot_1.*
- *$mwot_1$ and wot_1 have the same source and target states, so $\forall x.(s', t'_x | Pred_{s',t'_x}) \in m\mathcal{R}$ is preserved.*

and then:

$$\forall otmvars(mwot(M'_1)).\Big\{Pred_{s,t} \wedge Pred_1 \implies \bigvee_{x\in X}\Big[\exists\ otmvars(mwot_{2x}(M'_{2x})).$$
$$\Big(\forall j \in J.\gamma_j = \gamma_{jx} \wedge Pred_{2x} \quad \wedge \alpha_1 = \alpha_{2x} \wedge Pred_{s',t'_x}\{\!\{Post_1 \uplus Post_{2_x}\}\!\}\Big)\Big]\Big\} \tag{1}$$

Now remark that Equation (1) is universally quantified on all meta variables of $mwot_1$. Apply σ_1 to specialize the values of M'_1 we get:

$$\forall otvars(wot_1).\Big\{Pred_{s,t} \wedge Pred_1\{\!\{\sigma_1\}\!\} \implies \bigvee_{x\in X}\Big[\exists\ otmvars(mwot_{2x}(M'_{2x})).$$
$$\Big(\forall j \in J.\gamma_j\{\!\{\sigma_1\}\!\} = \gamma_{jx} \wedge Pred_{2x} \wedge \alpha_1\{\!\{\sigma_1\}\!\} = \alpha_{2x} \ \wedge Pred_{s',t'_x}\{\!\{Post_1\{\!\{\sigma_1\}\!\} \uplus Post_{2_x}\}\!\}\Big)\Big]\Big\} \tag{2}$$

Equation (2) still contains meta variables from M'_{2x}, so let us transform the conjunction by applying all possible substitutions $\sigma_2 \in \Sigma_2$, we get:

$$\forall otvars(wot_1).\Big\{Pred_{s,t} \wedge Pred_1\{\!\{\sigma_1\}\!\} \implies$$
$$\bigvee_{x\in X,\sigma_2\in\Sigma_2}\Big[\exists\ otvars(wot_{2x\ \sigma_2}).\Big(\forall j \in J.\gamma_j\{\!\{\sigma_1\}\!\} = \gamma_{jx}\{\!\{\sigma_2\}\!\} \tag{3}$$
$$\wedge\ Pred_{2x}\{\!\{\sigma_2\}\!\} \wedge \alpha_1\{\!\{\sigma_1\}\!\} = \alpha_{2x}\{\!\{\sigma_2\}\!\} \ \wedge Pred_{s',t'_x}\{\!\{Post_1\{\!\{\sigma_1\}\!\} \uplus Post_{2_x}\{\!\{\sigma_2\}\!\}\}\!\}\Big)\Big]\Big\}$$

which is the proof condition we are looking for.

Conclusion: Expansion of meta OAs preserves weak bisimulation; this concludes the proof of Theorem 3.7.

Running Example: Meta Weak Bisimulation: The Specification and Implementation of the running example should have a Weak FH-Bisimulation relation, meaning each OT of each OA must be *covered* by a WOT of the other system (in principle a set of WOT, but a single one is sufficient here). Indeed they are equivalent by (meta) Weak FH-Bisimulation, as can be sketched here:

The relation $\mathcal{R}$, Table 1 that is the candidate for our meta weak bisimulation relation.

Table 1: Bisimulation relation Triples of running example

Spec state	Impl state	Predicate
b0	000	True
b0	202	True
b1	100	`b_msg = s_msg` $\wedge$ `b_ec = s_ec`
b1	210	`b_msg = m_msg` $\wedge$ `b_ec = m_ec`
b1	220	`b_msg = s_msg` $\wedge$ `b_ec = s_ec`
b1	201	`b_msg = r_msg` $\wedge$ `b_ec = r_ec`

We give an example here, and prove how $WS_3(n)$ is covered by meta-WOT $WI_3(n')$.

$WS_3(n)$ has meta-Variable n. It is a self loop transition, which makes `b_ec` increase by 1 for each loop.

$$WS_3(n) = \frac{\{\texttt{P}\mapsto\texttt{p-send(m)}\}, True, (\texttt{b\_ec} \leftarrow n, \texttt{b\_msg} \leftarrow \texttt{m})}{b0 \xrightarrow{\texttt{in(m)}} b1}, n \geqslant 0$$

$WI_3(n')$ has meta-Variable n'. There is a loop transition path, $100 \rightarrow 210 \rightarrow 220 \rightarrow 100$, which makes s_ec increased by 1 for each loop.

$$WI_3(n') = \frac{\{\texttt{P}\mapsto\texttt{p-send(m')}\}, True, (\texttt{s\_msg} \leftarrow \texttt{m}', \texttt{s\_ec} \leftarrow n')}{000 \xrightarrow{\texttt{in(m')}} 100}, n' \geqslant 0$$

The predicates in the corresponding Triples are $Pred_{b0,000} = True$, $Pred_{b1,100} =$ (`b_msg = s_msg`$\wedge$ `b_ec = s_ec`).

Then, the proof obligation

$$\forall otmars(mwot).\Big\{Pred_{s,t} \wedge Pred_{mwot} \Longrightarrow \bigvee_{x\in X}\Big[\exists otmvars(mwot_x).$$

$$\Big(\forall j \in J_x.(\beta_j)^{\nabla} = \gamma_{jx} \wedge Pred_{mwot_x} \ \wedge \alpha = \alpha_x \wedge Pred_{s',t'_x}\{\!\{Post_{mwot} \uplus Post_{mwot_x}\}\!\}\Big)\Big]\Big\}$$

will be (as there are local variables `m` and `m'` in $WS_3(n)$ and $WI_3(n')$):

$$\forall \texttt{m}, n.\Big\{True \wedge True \ \Longrightarrow \exists\, \texttt{m}', n'.\Big(\texttt{p-send(m)} = \texttt{p-send(m')} \wedge True \wedge \ \texttt{in(m)=in(m')}$$

$$\wedge(\texttt{b\_msg = s\_msg} \ \wedge \ \texttt{b\_ec = s\_ec})\{\!\{(\texttt{b\_msg} \leftarrow \texttt{m}, \texttt{b\_ec} \leftarrow n) \uplus (\texttt{s\_msg} \leftarrow \texttt{m}', \texttt{s\_ec} \leftarrow n')\}\!\}\Big)\Big\}$$

Simplified as:

$$\forall \texttt{m}, n. \exists \texttt{m}', n'. \texttt{p-send(m)} = \texttt{p-send(m')} \wedge \texttt{in(m)=in(m')} \wedge \texttt{m} = \texttt{m}' \wedge n = n'$$

$\mathtt{m}$ and $\mathtt{m}'$ have the same domain; n and n' have the same domain also. So we can choose $\mathtt{m}' = \mathtt{m} \wedge n' = n$ and the proof obligation is a tautology. So, $WS_3(n)$ is covered by $WI_3(n')$.

3.4 Strategies for Weak Bisimulation Checking

This sub-section will discuss two ideas to check weak bisimulation between two OAs, 1) constructing complete WOAs before checking or 2) searching and building weak transitions on-the-fly on demand.

3.4.1 Construction of complete WOA

Construct complete WOAs (meta-WOAs in most cases) from given OAs before checking the weak bisimulation relations.

All OAs we discuss in this paper are generated from open pNets (as defined in [9, 20, 10]), but this could as well apply to OAs describing the behavior of other calculi or languages with explicit data, like Value-passing CCS [19] for example. According to the hierarchy architecture of pNets, the actions of sub-pNets or pLTS are internal and non-observable excepted those in the top-level synchronization vectors. With the growth of the hierarchy, the OA generated from a whole pNet will have a significant quantity of non-observable actions.

OA is a kind of directed graph, whose vertices are states and transitions are edges. These edges, transitions with non-observable actions, may lead to many loops that can be applied infinitely many times.

In the construction of WOA, we need to apply construction Rules, especially the rule **WT3**, as much as possible, until for every Triple of source state, target state, and action (only for these observable). Then the meta-WOTs represent all WOTs.

There will be two difficulties to do so:

- Calculating Self-Loop meta-WOT with non-observable actions.
- Several Self-Loop meta-WOTs with non-observable actions at the same state.

We use a simple example to illustrate these difficulties. It's a part of our use-case Specification OA, Figure 8(A), but changed a little to illustrate our point. Specifically, we split SS_4 into ot_1 and ot_2. In other words, the increment function of error count ($\mathtt{b\_ec} \leftarrow \mathtt{b\_ec + 1}$) is replaced by two magic functions $f(x)$ and $g(x)$ (the unsafe medium increases the error count by two distinct magic functions).

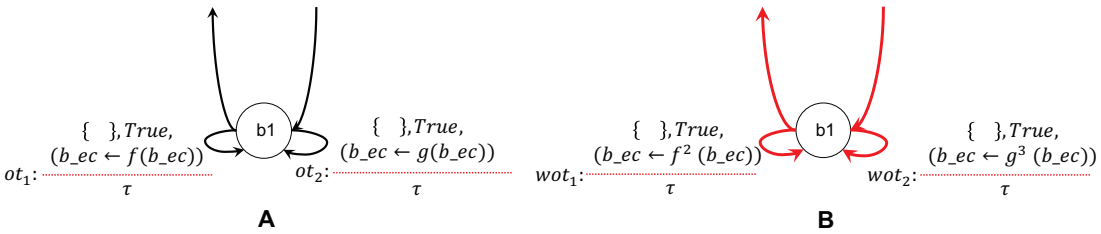


Figure 8: Build WOT, **(A)** is a part of the Specification OA, **(B)** is a part of some possible intermediate step while generating the Specification WOA.

In Figure 8(B) we build WOTs, from ot_1 and ot_2 using rule **WT2**. As they have non-observable actions, they can repeat many times. Say, wot_1 repeats two times and wot_2 repeats three times. According to $ot_1.Post = (x \leftarrow f(x))$, the result of $f(x)$ will be the input of next iteration of applying **WT3**. Namely $x \leftarrow f^2(x)$ on one side, $x \leftarrow g^3(x)$ on the other side.

It's not a difficult job to build wot_1 and wot_2, even by hand. But, with the increment of repetition, it will be harder. Moreover, theoretically, on each of them **WT3** can be applied infinitely many times.

$$x \leftarrow f(f(f(...f(x)))) : \qquad \forall n \geqslant 1, x \leftarrow f^n(x)$$

Then we will have

$$wot_1(n).Post = (\texttt{b\_ec} \leftarrow f^n(\texttt{b\_ec}))$$

It will be not easy to infer (by an algorithm) what exactly it is, because the $f(b\_ec)$ can be some complex mathematical function in practice. So, here is one of the difficulties in figuring out the Self-Loop meta-WOT with non-observable action.

Then, presuming we have figured out the two Self-Loop meta-WOTs, $mwot_1(n_1)$ and $mwot_2(n_2)$ in some way, by hand maybe. It is still not easy to compute all the possible Self-Loop transitions whose source and target states are both $b1$ with non-observable action, τ. Because of the non-observable action property, $mwot_1(n_1)$ and $mwot_2(n_2)$ can be applied in any order. Formally we have an infinite transition sequence TS encoding all possible OT sequences in state $b1$, expressed as the regular expression:

$$TS \leftarrow (ot_1|ot_2)^*$$

In section 5.2.3 we shall define formally a notion of *independence* to deal with this question. The idea is that if $mwot_1(n_1)$ and $mwot_2(n_2)$ have some intricate interactions, then finding out expected Self-Loop WOT by assignments of meta-Variables is not enough. We need to try all possible orders of these Self-Loop meta-WOTs.

Besides these two difficulties, the construction of the complete WOA is not an efficient way to check weak bisimulation. Because it may happen that in the procedure of building WOA, it is already impossible to find out some expected WOT for the corresponding OT and relation Triple, which means it may be already sufficient to prove the unsatisfiability of weak bisimulation, but we still need to build the rest of WOA in vain.

In summary, it is not a good idea to build complete WOAs or meta-WOAs before checking weak bisimulation relations, as this construction has several ways of being infinite.

3.4.2 Construction of WOT on-the-fly

Focusing on checking weak bisimulation relations, finding out one specific WOT on-the-fly, corresponding to the considered OT and relation Triples, obeying directly the bisimulation definition, is more efficient.

Our goal here is to check weak bisimulation relation given two input OAs and a candidate relation R defined as a TripleSet. We know the source state and the target states for each target OT, and then we can search a transition sequence for building WOTs from this source state on demand. To avoid the difficulties mentioned in the previous section, we shall set an upper bound to the search algorithm. If one of the search branches is out of bound, the algorithm ends this search branch and starts another until finding one transition sequence with the expected target state. Then we build a WOT according to the transition sequence and check weak bisimulation relation. It is more efficient and practicable than the first idea.

More detail will be described in the next section.

4 Checking Algorithm

This section describes the algorithm for building WOT on-the-fly when checking the Weak FH-Bisimulation relation with given two OAs and $\mathcal{R}$ as Triples. We have two search methods, Bounded DFS and Bounded BFS, for different system situations. Moreover, a brief talk about how to design (guess) the bound.

This algorithm depends on the Satisfiability Modulo Theories (SMT), which develops from the Boolean satisfiability problem (SAT). Satisfiability of the proof obligation, which is built from a transitions pair and the corresponding relation Triples, is an SMT problem. We choose Z3 as the SMT-solver engine with potential determined responses, TRUE, FALSE, and non-determined responses, TIMEOUT, UNKNOWN. Z3 is not good at dealing with recursive data-types and functions, which often leads to non-determined responses.

The algorithm is semi-decidable when the Z3 returns a determined response, and the bound should be large enough for searching the expected WOT. That means when the algorithm gives a positive response, and it guarantees the bisimulation fact. However, when the algorithm gives 'Failed' response, there are several possible reasons: the bound is too strong for searching WOT, or the Z3 returns a non-determined response, or maybe the two OAs do not have the bisimulation fact, or target OT can be covered by a set WOT.

4.1 Main Body of the Algorithm

The main function is CHECKTRIPLES. In this function, we traverse all the Triples and build corresponding WOTs from one OA on demand and check whether all of them meet the definition of Weak StrFH-Bisimulation. After this, we should build WOTs from the other OA for the symmetrical property and process the same procedure. The algorithm is the same for both. Thus we only show the first direction.

We have input with two OAs, OA_1 and OA_2, and Triples set $\mathcal{R}$. First, we build a new WOA, woa_2, from OA_2 only to apply **WT2**. Let $(s, t|Pred_{s,t})$ become the Triple we are checking now. For each OT starts from s (say, *ot* from s to s'), we find out these Triples with their first item as s'. From each of these Triples, say, $(s', t'|Pred_{s',t'})$, we try to build a WOT from t to t' by the bounded search algorithm by calling $VERIFYWOT$.

$VERIFYWOT$ is a bounded DFS(BFS) algorithm that has bound of the depth of search. Once we get one expected WOT, we will construct a corresponding proof obligation (actually, the negation of the proof obligation) to check the satisfiability by SMT engine, Z3.

When Z3 returns $UNSAT$ (it means the proof obligation is a tautology), that means current OT (variable *ot*) is covered, and we continue this process for all OTs whose source state is s. If all OTs are covered, then we can conclude that $(s, t|Pred_{s,t})$ fits the definition of Weak FH-Bisimulation and go for the next Triple.

If all the Triples pass this check, it means all the Triples in $\mathcal{R}$ meet the requirements; thus, we can say $\mathcal{R}$ is a Weak FH-Bisimulation between OA_1 and OA_2 and return true.

4.2 Bounded Searching

This sub-section describes the details of the bounded search algorithm, which builds WOTs on-the-fly on demand. It is also the significant part of the whole checking algorithm. There are two kinds of searching styles, DFS and BFS. They are almost the same but on the different searching priorities of branches. We only show the pseudocode of Bounded DFS in the paper.

For the input open transition *ot*, Triples $Triple = (s, t|Pred_{s,t})$, $Triple' = (s', t'|Pred_{s',t'})$, and the weak OT *wot*, without losing generality, we have: *ot* is the OT which should be covered

Algorithm 1 Checking Weak Bisimulation between Two OAs by Generate WOT on-the-fly

input: OA_1 and OA_2 two open automata, where $OA_1 = \langle\!\langle J_1, \mathcal{S}_1, init_1, V_1, \mathcal{T}_1 \rangle\!\rangle$ and $OA_2 = \langle\!\langle J_2, \mathcal{S}_2, init_2, V_2, \mathcal{T}_2 \rangle\!\rangle$. $\mathcal{R}$ is a set of Triples, where $\mathcal{R} = \{(s, t | Pred_{s,t}) | s \in \mathcal{S}_1, t \in \mathcal{S}_2\}$
output: A variable with two possible values: $True$ means the input OAs have relation R; $Failed$ means the algorithm failed to decide the relation.

```
1  function CHECKTRIPLES(OA1, OA2, R)
2      woa2 ← apply WT2 on OA2
3      for each Triple (s, t|Pred_{s,t}) in R
4          for each Open Transition ot in OA1 from state s to s'
5              res ← Failed
6              for each Triple with first element as s', says
7              Triple' = (s', t'|Pred_{s',t'})
8                  wot_τ ← apply WT1 on t
9                  res ← VERIFYWOT(woa2, ot, Triple, Triple', wot_τ)
10                 if res is Failed then
11                     return res
12                 else if res is True then
13                     break
14                 end if
15             end for
16             if res!= True then
17                 return res
18             end if
19         end for
20     end for
21     return res
22 end function
```

by a set of WOT according to Definition 2.11;

$$ot = \frac{\beta_j^{j \in J'}, Pred_{ot}, Post_{ot}}{s \xrightarrow{\alpha} s'}$$

wot is an old WOT which was built in the previous pass, say it ends in t_{mid}. But in the first call of searching algorithm, wot is a Self-Loop WOT_τ, which means $t = t_{mid}$.

$$wot = \frac{\gamma_j^{j \in J'}, Pred_{wot}, Post_{wot}}{t \xRightarrow{\alpha'} t_{mid}}$$

wot' is one of the WOTs that has been explored in the current DFS iteration with source state as same as the target state of wot, t_{mid}.

$$wot' = \frac{\gamma_{jx}^{j \in J'}, Pred_{wot_x}, Post_{wot_x}}{t_{mid} \xRightarrow{\alpha''} t'_x}$$

Then we should *Combine* wot and wot' (apply **WT3** on them). However, we can only have at most one transition with observable action according to Rule **WT3**. So, when α' is non-observable or α'' is non-observable, we can perform the Combine process, and $newwot$ is the result.

Algorithm 2 Build WOT on demand to check $\mathcal{R}$ with target OT

input: *woa* is a WOA, *ot* (from s to s') is the OT needs to be covered , $Triple$ $((s,t|Pred_{s,t}))$ is the Triple in $\mathcal{R}$ which need to be checked, $Triple'$ $((s',t'|Pred_{s',t'}))$ is the corresponding Triple of *ot*, *wot*(from t to t_{mid} with action α') is a WOT .
output: A variable with two possible values: $True$ means the current proof obligation is Tautology; $Failed$ means the algorithm Failed.

```
1  function VERIFYWOT(WOA, OT, Triple, Triple', wot)
2      if length(wot) > LIMITS then
3          return Failed
4      end if
5      res ← Failed
6      for each wot' in woa from t_mid , says to t'_x
7          α'' ← action of wot'
8          if (α' is non−observable) or (α'' is non−observable) then
9              newwot ← Combine(wot, wot')
10             if t'_x = t' then
11                 negateOb ← generate negation of proof obligation
12                 use SMT−solver to check negateOb's satisfiability
13                 if SMT−solver returns UNSAT then
14                     return True
15                 end if
16             end if
17             featureRes ← VERIFYWOT(woa, ot, Triple, Triple', newwot)
18             if featureRes is True then
19                 return True
20             end if
21         end if
22     end for
23     return res
24 end function
```

$$newwot = \frac{\gamma'^{j\in Jx}_{jx}, Pred'_{wot_x}, Post'_{wot_x}}{t \overset{\alpha_x}{\Longrightarrow} t'_x}$$

If $t'_x = t'$, that means *newwot* could be a potential expected WOT for the target OT. We generate a proof obligation by definition:

$$Pred_{s,t} \wedge Pred_{ot} \implies \left(\forall j \in J_x.\beta_j = \gamma'_{jx} \wedge \ Pred'_{wot_x} \wedge \ \alpha = \alpha_x \wedge Pred_{s',t'}\{\!\{Post_{ot} \uplus Post'_{wot_x}\}\!\}\right) \quad (4)$$

And its negation:

$$Pred_{s,t} \wedge Pred_{ot} \wedge \left(\exists j \in J_x.\beta_j \neq \gamma'_{jx} \vee \ \neg Pred'_{wot_x} \vee \ \alpha \neq \alpha_x \vee \ \neg Pred_{s',t'}\{\!\{Post_{ot} \uplus Post'_{wot_x}\}\!\}\right) \quad (5)$$

Thus, we can check the satisfiability of this Formula 5 with SMT-solver (Z3). If this Formula 5 is unsatisfiable, it means the original proof obligation is a tautology. In other words, we already found one WOT which matches the expected OT, and the function returns $True$ and

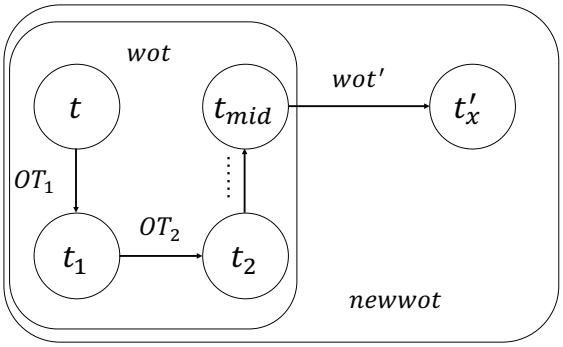


Figure 9: Weak FH-Bisimulation Checking Algorithm

stop. Otherwise, we should continue to explore an extended transition sequence with the DFS algorithm. Only when no more *newwot* can be constructed within the given bound and proof obligations never be tautology , then the algorithm return $Failed$.

The checking of the proof obligation is strongly dependent on SMT-solver. As long as the SMT-solver can determine all the generated proof obligations, we can say our algorithm can determine the weak bisimulation problem on the given automata and TripleSet. If there is a proof obligation that SMT-solver cannot solve, our algorithm will skip it and continue searching.

Every time processing the function, we should check the length of *wot* to prevent exceeding the bound. If the current depth is over bound, we stop the current iteration and return $Failed$, which means we failed to find an expected WOT in the current branch with the bound.

In short, if we find any *newwot* that makes Formula 4 a tautology, the final answer is $True$; otherwise we get $Failed$.

4.3 Characteristics of the bounded search algorithm

In this sub-section, we will discuss the termination and correctness of the algorithm, and the difference between these two styles of searching. Moreover, we give some advice how to guess the bound.

4.3.1 Termination

The weak bisimulation relation R, as Triples, is finite. Furthermore, the OAs are also finite, which means the number of transitions from every state is finite. So, the "for" loop in "CHECK-TRIPLES" is finite. We have a bounded depth of searching. For each proof obligation, as long as SMT-solver can decide it, it will terminate and give a result.

In conclusion, as long as we input a finite TripleSet and two finite OAs, and the case is decidable (Z3 returns determinate response), our algorithm will terminate.

4.3.2 Correctness

The main algorithm, $CHECKTRIPLES$, traverses all the given Triples and all corresponding OTs for the bounded DFS algorithm. It searches for all possible transitions sequences, within the bound, to build WOT and check whether the proof obligation satisfies the definition by SMT-engine. Once Z3 returns $UNSAT$, which means we found the expected WOT. If each OT

has a corresponding WOT found, then the weak bisimulation relation holds. The correctness is guaranteed.

4.3.3 Variants

These two search methods both have their advantages and weaknesses. Bounded DFS suits large-scale automata with numerical transitions for each global state; Bounded BFS has a higher possibility of finding expected WOT faster, but with possible greater demand on memory.

Let us analyze the details. Bounded DFS digs into one branch until finding an expected WOT or exceed the limits. But the expected WOT may hide in the later branch, which means the DFS needs to explore all the earlier branches until exceeding the bound. It may be not efficient enough but needs less memory. Because every time the function is called, only one searching path is needed as the parameter.

As for Bounded BFS, it explores all possible branches at each level of searching, which means it maybe find the solution faster if the expected path does not have a large length. Nevertheless, it also needs all current explored paths as parameters. It may need more memory resources for them.

As for the intuition about the length of the expected path, we could analyze the Post from given OAs. It can be very different depending on practical instances.

4.3.4 Guess Bound

One of the difficulties is guessing a suitable bound for these two algorithms. If the bound is too low, both of them may not be able to find the solution; But with a large bound, especially for the Bounded DFS, it maybe stacks into hopeless branches and waste plenty of time.

Here is a piece of good advice for experts to guess the bound: pay attention to the Pred and Post of OAs. In most cases, the specification OA would not be too complicated (compared to the Implementation OA), so with Pred and Post's help, we can guess a reasonable bound.

If it is still hard to guess, set the bound as the number of all OA transitions. It gives the possibility to go through all the transitions at least once to find the solution.

4.3.5 Bounded BFS

This method is similar to Bounded DFS but with a breadth-first search.

Algorithm 3 Build Weak Open Transition on demand to check $\mathcal{R}$ with target Open Transition

input: *woa* is a WOA, ot(from s to s') is the OT needs to be covered, $Triple((s,t|Pred_{s,t}))$ is the Triple in $\mathcal{R}$ which need to be checked, $Triple'((s',t'|Pred_{s',t'}))$ is the corresponding Triple of ot, *wotSet* is a WOT Set

output: a variable with two possible value: $True$ means the current the Proof Obligation is Tautology, $Failed$ means the Algorithm Failed.

```
1  function VERIFYWOT(woa, ot, Triple, Triple', wotSet)
2      if length(wot) > LIMITS then
3          return Failed
4      end if
5      res ← False ; newWOTSet ← ∅
6      for wot in wotSet from $t$ to $t_{mid}$ with action $\alpha'$
7          for each wot' in woa starts from t_mid
8              α'' ← Action of wot'
9              if (α' is non-observable) or (α'' is non-observable) then
10                 newWOT ← Combine(wot, wot')
11                 newWOTSet ← newWOTSet ∪ {newWOT}
12             end if
13         end for
14     end for
15     if newWOTSet is ∅ then
16         return res
17     end if
18     for newWOT in newWOTSet
19         if newWOT.targetState is t' then
20             negateOb ← generate negation of proof obligation
21             use SMT-solver to check if negateOb is satisfiable
22             if SMT-solver returns UNSAT then
23                 return True
24             else
25                 continue;
26             end if
27         end if
28     end for
29     res ← VERIFYWOT(woa, ot, Triple, Triple', newWOTSet)
30     return res
31 end function
```

Combine is the function which apply the Rule **WT3** with details.

Algorithm 4 Build a new Weak Open Transition by Rule **WT3**

input: two weak open transition wot_1 and wot_2, where $wot_1.target = wot_2.source$
output: a new wot build by inputs or a emptyset

```
1  function Combine(wot_1, wot_2)
2      wot ← a new weak open transition
3      wot.Pred ← wot_2.Pred{{wot_1.Post}}
4      if wot.Pred = False then:
5          return ∅
6      end if
7
8      if wot_1.Action and wot_2.Action are observable then
9          return ∅
10     else if wot_1.Action is observable then
11         wot.Action ← wot_1.Action
12     else if wot_2.Action is observable is observable then
13         wot.Action ← wot_2.Action{{wot_1.Post}}
14     else
15         wot.Action ← τ
16     end if
17
18     wot.γ̄ ← an empty set for sequence of Holes Actions
19     for j ∈ 𝕁
20         wot.γ̄_j ← wot_1.γ̄_j ⊎ wot_2.γ̄_j{{wot_1.Post}}
21     end for
22
23     wot_2.Post is (x_k ← e_k)^{k∈K}
24     wot_1.Post is (x'_{k'} ← e'_{k'})^{k'∈K'}
25     wot.Post ← (x_k ← e_k{{(x'_{k'} ← e'_{k'})^{k'∈K'}}})^{k∈K} ∪ (x'_{k'} ← e'_{k'})^{k'∈K''}
26     wot.source ← wot_1.sourceState
27     wot.target ← wot_2.targetState
28
29     return wot
30 end function
```

5 Minimization and Reduction

In this section, we discuss minimization of open automata, and propose a framework to efficiently reduce then while preserving Weak FH-Bisimulation:

- We first prove that any OA can be reduced to a one state automaton while preserving strong FH-Bisimulation, and argue that minimization in that sense has no practical interest.
- Then we introduce a method to define OA reduction using some kind of pattern based rules, define a (weak) simulation notion for open transition, and prove that if such a rule 'locally preserves' Weak FH-Bisimulation, then it preserves Weak FH-Bisimulation for the whole OAs.
- Finally we define 3 such reduction rules (*tau-Merging*, *Forward Merging*, and *Backward Merging*), prove that *tau*-Merging respect the local weak simulation, and show that these

three rules are sufficient to reduce the Implementation OA of our running example to a small and simple OA weakly bisimilar to the Specification.

5.1 Open Automata minimization

In this sub-section, we will introduce two styles of OA and a special Minimization, i.e., *One State Minimization*.

5.1.1 Two Styles of Open Automata

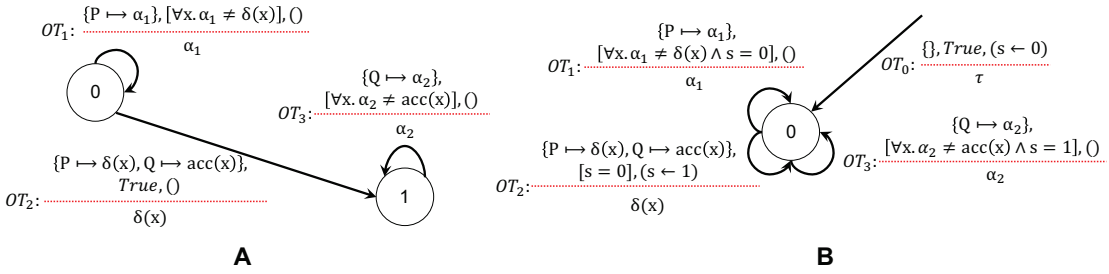


Figure 10: Two Styles Examples from Lotos, **(A)** is the Action-Oriented Example, **(B)** is the Data-Oriented Example.

Here is a variant example using the Enable operator from Lotos. The only difference from original Enable operator is turning the δ action into an observable action. We use pNets to construct this example [10], and get OAs from it. There are two different styles: the action-oriented style Figure 10A, and the data-oriented style Figure 10B.

5.1.2 One State Minimization

Theoretical, any OA can be compressed in a data-oriented style with one single state with all Self-Loop transitions by adding one extra variable representing the different stages of the system, the variable s in Figure 10B. It is the so-called *One State Minimization*.

Definition 5.1 (One State Minimization (OSM)) *Let $A = \langle\!\langle J, \mathcal{S}, V, s_0, \mathcal{T} \rangle\!\rangle$. With the following procedure, we can minimize A's number of states and get an equivalent open automaton $OSM(A) = \langle\!\langle J, \{ms\}, ms, V\{\!\{\sigma_V\}\!\} \cup \{ID\}, \mathcal{T}' \rangle\!\rangle$, where* ms *is the merged state. The OSM transformation works in the following way:*

Let ms *be a fresh state name, and ID be a new variable that will encode the names of the original states, with initial value s_0. $\sigma_V = \bigcup_{v \in V} (v \leftarrow v')$ is a substitution that replaces all variables with fresh variables, to ensure the new automaton has no variable name conflicts with the original one.*

Equivalently, we shall use the logical counterpart of σ_V as a predicate $\left[\bigwedge_{v \in V} v = v' \right]$.

For each open transition $ot = \dfrac{\beta_j^{j \in J}, Pred, Post}{s \xrightarrow{\alpha} s'}$ construct $OSM(ot) = \dfrac{\beta_j'^{j \in J}, Pred', Post'}{ms \xrightarrow{\alpha'} ms}$

with:

$$\beta_j'^{j\in J} = \beta_j^{j\in J}\{\!\{\sigma_V\}\!\}$$

$$Pred' = Pred\ \{\!\{\sigma_V\}\!\} \wedge [ID = s]$$

$$Post' = Post\ \{\!\{\sigma_V\}\!\} \cup \{ID \leftarrow s'\}$$

$$\alpha' = \alpha\ \{\!\{\sigma_V\}\!\}$$

Then, $\mathcal{T}' = \{OSM(ot)\}^{ot\in\mathcal{T}}$.

Remark that the One State Minimization transformation only minimizes the number of states but keeps the same number of transitions. And we have the theorem:

Theorem 5.2 *For any OA, OA and OSM(OA) are strongly FH-Bisimilar.*

Proof 5.3
Given any open automaton, $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T}\rangle\!\rangle$*, with One State Minimization Definition 5.1, we construct a new automaton* $OSM(A) = A' = \langle\!\langle J, S' = \{ms\}, ms, V\{\!\{\sigma_V\}\!\} \cup \{ID\}, \mathcal{T}'\rangle\!\rangle$*, that has only one state by construction.*

Intuitively, the Strong Bisimulation TripleSet we need is quite simple, as shown in Table 2 (including the initial condition $Pred_{init}$*).*

Table 2: Relation Triples for One State Minimization

Origin Automaton	Minimized Automaton	Predicate
s_0	ms	$[\bigwedge_{v\in V} v = v'] \wedge ID = s_0 \wedge Pred_{init}$
s_1	ms	$[\bigwedge_{v\in V} v = v'] \wedge ID = s_1$
...	...	...
s_n	ms	$[\bigwedge_{v\in V} v = v'] \wedge ID = s_n$

Part 1:
For any OT in A, named ot_1*:* $ot_1 = \overset{\overline{\beta}, Pred, Post}{\overset{\alpha}{s_s \longrightarrow s_t}} \in \mathcal{T}$
there exists a corresponding ot_1' *in* A'*,* $ot_1' = \overset{\overline{\beta'}, Pred', Post'}{\overset{\alpha'}{ms \longrightarrow ms}} \in \mathcal{T}'$
According to Definition 5.1, we have (recall J is the set of Hole indexes):

$$\forall j \in J.\ \beta_j' = \beta_j\ \{\!\{\sigma_V\}\!\}$$

$$Pred' = Pred\{\!\{\sigma_V\}\!\} \wedge [ID = s_s]$$

$$Post' = Post\{\!\{\sigma_V\}\!\} \cup \{ID \leftarrow s_t\}$$

$$\alpha' = \alpha\{\!\{\sigma_V\}\!\}$$

Or equivalently:

$$[\bigwedge_{v\in V} v = v'] \implies \forall j \in J.\ \beta'_j = \beta_j\ \{\!|\sigma_V|\!\} \wedge Pred\{\!|\sigma_V|\!\} = Pred \wedge Post\{\!|\sigma_V|\!\} = Post \wedge \alpha' = \alpha$$

Now we construct the proof obligation of Strong Bisimulation:

$$Pred_{s_s,ms} \wedge Pred \implies \forall j \in J.\ \beta_j = \beta'_j \wedge\ Pred' \wedge \alpha = \alpha' \wedge Pred_{s_t,ms}\{\!|Post \uplus Post'|\!\}$$

That gives us:

$$[\bigwedge_{v\in V} v = v' \wedge ID = s_s] \wedge Pred \implies ([\bigwedge_{v\in V} v = v' \wedge ID = s_t]\{\!|Post \uplus (Post\{\!|\sigma_V|\!\} \uplus \{ID \leftarrow s_t\})|\!\})$$

We know from Definition 5.1 that ID is new variable ($ID \notin V$), so $ID \notin vars(Post)$ and the final substitution,$\{\!|ID \leftarrow s_t|\!\}$ transforms $[ID = s_t]$ into $[s_t = s_t]$. The proof obligation is a tautology, i.e., ot'_1 covers ot_1.

So, any OT from s_s in $\mathcal{T}$ can be covered by a set of transitions from ms in $\mathcal{T}'$.

Part 2:

Now, we need to prove for the reverse direction, i.e., for any ot' in $\mathcal{T}'$ there is at least one ot from $\mathcal{T}$ cover it. We remark that the triples in Table 2 establish a one-to-one relation between each state s_i in the original automaton, and the value of ID in the corresponding predicate.

So for any transition OT in A', named ot'_2:

$$ot'_2 = \frac{\overline{\beta'}, Pred'_2, Post'_2}{ms \xrightarrow{\alpha_2} ms} \in \mathcal{T}'$$

there is exactly one ot in $\mathcal{T}$:

$$ot_2 = \frac{\overline{\beta}, Pred_2, Post_2}{s_s \xrightarrow{\alpha_2} s_t} \in \mathcal{T}$$

such that $Pred'_2$ contains $[ID = s_s]$ and $Post'_2$ contains $\{\!|ID \leftarrow s_t|\!\}$ and $ot'_2 = OSM(ot_2)$.

And by construction:

$$\forall j \in J.\ \beta'_j = \beta_j\{\!|\sigma_V|\!\}$$

$$Pred'_2 = Pred_2\{\!|\sigma_V|\!\} \wedge [ID = s_s]$$

$$Post'_2 = Post_2\{\!|\sigma_V|\!\} \cup \{ID \leftarrow s_t\}$$

$$\alpha'_2 = \alpha_2\{\!|\sigma_V|\!\}$$

According to Table 2, we have Triples:

$$(s_s, ms, Pred_{s_s,ms} = ([\bigwedge_{v\in V} v = v'] \wedge ID = s_s))$$

$$(s_t, ms, Pred_{s_t,ms} = ([\bigwedge_{v\in V} v = v'] \wedge ID = s_t))$$

Then the corresponding proof obligation:

$$Pred_{s_s,ms} \wedge Pred'_2 \implies \forall j \in J.\beta'_j = \beta_j \wedge Pred_2 \wedge \alpha'_2 = \alpha_2 \wedge Pred_{s_t,ms}\{\!\{Post'_2 \uplus Post_2\}\!\}$$

gives us:

$$\Big[\bigwedge_v^V v = v' \wedge ID = s_s\Big] \wedge \Big[Pred_2\{\!\{\sigma_V\}\!\} \wedge ID = s_s\Big] \implies \forall j \in J.\beta'_j = \beta_j \wedge Pred_2 \wedge \alpha'_2 = \alpha_2$$

$$\wedge \Big(\Big[\bigwedge_v^V v = v' \wedge ID = s_t\Big]\{\!\{(Post_2\{\!\{\sigma_V\}\!\} \cup \{ID \leftarrow s_t\}) \uplus Post_2\}\!\}\Big)$$

The final substitution $\{\!\{ID \leftarrow s_t\}\!\}$ *transforms* $[ID = s_t]$ *into* $[s_t = s_t]$*. The proof obligation is a tautology, i.e.,* ot_2 *covers* ot'_2*.*

In summary, A and $OSM(A)$ *are equivalent for the strong FH-Bisimulation relation using the Tripleset in Table 2.*

5.2 Open Automata Reduction

In this sub-section, we give out the intuition and details for Reduction. The key parts are *Merging Transitions* , *Merging Principles* (patterns) and *Local FH-Bisimulation.*

5.2.1 Intuition for Reduction

Given an open automaton, the purpose of reduction is to build a new automaton with a smaller size on states and transitions.

The output of the Reduction is still an open automaton, but with less transitions and states. That means we need new rules for merging transitions and combine equivalent (maybe with Predicates) states. These rules will be illustrated later. The minimized open automaton is weakly bisimular with the original one, in the sense of definition [2.11].

Reduction gives a smaller equivalent OA, while Minimization should give the smallest (or a representative of the class of smallest) OA, in the number of states, and potentially in the number of transitions. However, given Theorem 5.2, full Minimization has no interesting meaning in practice, as any automaton can be reduced to a strongly equivalent one state automaton.

However, Weak FH-Bisimulation can allow us to get more reductions in terms of transitions. We present an algorithm for the Reduction of OAs preserving Weak FH-Bisimulation, using three reduction rules based on graph patterns. These rules yield useful reductions merging both open transitions and states of OAs, and we show how this works in practice on our running example.

5.2.2 Merging Transitions

The intuitive idea of the reduction algorithm is quite simple: merging these transitions with non-observable actions with its next or previous transition.

But in the process, we have to make sure the merging result is a kind of special WOT, which can be transformed into OT by the reverse operation of **WT1**. Comparing the elements between WOT and OT, the difference is the Holes' actions. In an OT, each Hole can have at most one action or empty, while in a WOT each Hole can have a sequence of actions. So, we need to keep the result transition of our reduction transformation satisfying the definition of OT.

The rules of merging below are almost the same as rule **WT3** defining the construction of WOTs, but slightly modified to respect this constraint, formally defined as follows:

Definition 5.4 (Merging Transitions) *Let $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$ be an open automaton, from which we can build $woa = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{WT} \rangle\!\rangle$. We can merge transitions with the following rules:*

$$\frac{\overline{\gamma_1}, Pred_1, Post_1 \quad \overline{\gamma_2}, Pred_2, Post_2 \quad Pred = Pred_1 \wedge Pred_2\{\!|Post_1|\!\} \quad \overline{\gamma} = \overline{\gamma_1} \cup \overline{\gamma_2}\{\!|Post_1|\!\} \quad \forall i \in \overline{\gamma}, length(\gamma_i) \leqslant 1 \quad \alpha' = \alpha\{\!|Post_1|\!\}}{\overline{\gamma}, Pred, Post_2 \otimes Post_1} \textbf{MT1}$$

$$\frac{\overline{\gamma_2}, Pred_2, Post_2 \quad \overline{\gamma_3}, Pred_3, Post_3 \quad Pred = Pred_2 \wedge Pred_3\{\!|Post_2|\!\} \quad \overline{\gamma} = \overline{\gamma_2} \cup \overline{\gamma_3}\{\!|Post_2|\!\} \quad \forall i \in \overline{\gamma}, length(\gamma_i) \leqslant 1 \quad \alpha' = \alpha}{\overline{\gamma}, Pred, Post_3 \otimes Post_2} \textbf{MT2}$$

Basically, the procedure of Merging Transitions splits the WOT construction rule **WT3**. **MT1** merges a transition with non-observable action to one of its next transitions. It will be used in *Forward Merging* and *τ-Merging*. **MT2** merges a transition with non-observable action to one of its previous transitions. It is used in *Backward Merging*. Both rules can also be used at OT level: when we apply **MT1** or **MT2** on OTs, we first apply **WT1** on the OTs (turning them into WOTs) by default. Then the result WOT is such that the holes behaviors have a length limited to 0 or 1, so it can be turned back to an OT.

5.2.3 Transition Merging Principles

We shall specify the *Reduction* of open automata by a set of *Reduction rules*, in the form of patterns, matching a small set of states and transitions, that will be transformed (reduced) by the rule, under some semantic conditions on the transitions elements (Holes, predicates, and Posts). These conditions allow us to ensure the preservation of the semantics by reduction: the rules preserve weak bisimulation. Technically, we define a notion of local weak simulation between 2 transitions.

Before going into the details, it is crucial to stress the following principles:

- All non-observable actions (action terms with underline and τ) are equivalent no matter whether they are carrying variables.
- It's impossible to distinguish whether a Hole has a non-observable action or when it is idle in the transition.Formally, we shall assume in all the forthcoming definitions and proofs, that the OAs involved have the non-observability of *tau* actions property, as in Definition 2.6.

The conditions for transition simulation are similar to those for *coverage* of predicate because of the definition of Strong or Weak FH-Bisimulation. However, we consider the case where the correspondence between transitions is more straightforward than the general case: to deal with reduction expressed by rewriting rules we do not need the complexity of "covering" one transition

of one of the systems by several transitions of the other; all our merging rules are reducing transitions in a one-to-one manner. We formalize the corresponding (two-way) simulation relations between transitions in the following way:

Definition 5.5 (Transitions Weak Simulation) *Consider two open automata A_1 and A_2 which Variable sets are disjoint, and two open transitions ot_1 from A_1 and ot_2 from A_2, with the same set of active holes J, such that :*

$$ot_1 = \frac{\overline{\beta_1}, Pred_1, Post_1}{s_1 \xrightarrow{\alpha_1} s_1'}, \quad and \quad wot_1 = \frac{\overline{\gamma_1}, Pred_1, Post_1}{s_1 \xRightarrow{\alpha_1} s_1'}, \qquad \forall j \in J, \gamma_{1j} = (\beta_{1j})^{\nabla}$$

$$ot_2 = \frac{\overline{\beta_2}, Pred_2, Post_2}{s_2 \xrightarrow{\alpha_2} s_2'}, \quad and \quad wot_2 = \frac{\overline{\gamma_2}, Pred_2, Post_2}{s_2 \xRightarrow{\alpha_2} s_2'}, \qquad \forall j \in J, \gamma_{2j} = (\beta_{2j})^{\nabla}$$

We say that ot_1 and ot_2 simulate each other (or "ot_1 and ot_2 are similar"), and we write $ot_1 \overset{WS}{\Longleftrightarrow} ot_2$ (where 'WS' is the abbreviation of 'weak simulate'), when there exist two predicates $Pred_{s_1,s_2}$ and $Pred_{s_1',s_2'}$ such that the following two formulas are tautologies:

(1) $Pred_{s_1,s_2} \wedge Pred_1 \implies \forall j \in J.(\beta_{1j})^{\nabla} = \gamma_{2j} \wedge Pred_2 \wedge \alpha_1 = \alpha_2 \wedge Pred_{s_1',s_2'}\{\!|Post_1 \uplus Post_2|\!\}$

(2) $Pred_{s_1,s_2} \wedge Pred_2 \implies \forall j \in J.(\beta_{2j})^{\nabla} = \gamma_{1j} \wedge Pred_1 \wedge \alpha_2 = \alpha_1 \wedge Pred_{s_1',s_2'}\{\!|Post_2 \uplus Post_1|\!\}$

Then we extend point-wise the definition to sets of transitions:

Definition 5.6 (Transition Set Weak Simulation) *If there exists a bijective function Φ which works on two sets of transitions $\mathcal{T}, \mathcal{T}'$, where the domain of Φ is $\mathcal{T}$ and the co-domain is $\mathcal{T}'$, and $\forall t \in \mathcal{T}.t \overset{WS}{\Longleftrightarrow} \Phi(t)$. Then these two transition sets simulate each other, and we write: $\mathcal{T} \overset{WS}{\Longleftrightarrow} \mathcal{T}'$.*

In the Implementation OA of our running example, WI_1, WI_2, WI_τ of each state form a set of transitions, say $SFL_s = \{WI_1, WI_2, WI_\tau | of\ state\ s\}$. Then, we duplicate the Implementation OA, using ' to distinguish them. SFL_s and $SFL_{s'}$ are *Similar*. For example, $\{WI_1, WI_2, WI_\tau | of\ state\ 000\}$ is Similar with $\{WI_1, WI_2, WI_\tau | of\ state\ 100'\}$.

The reason why *Similar* transitions are essential is that, in some cases, we want to merge two neighboring states with *Similar* Self-Loop transitions (technically, we should duplicate the variable set to match the Definition 5.5). Because the order of these *Similar* transitions in transition sequence would not affect the semantic meaning except the order of the state, the semantic range of WOT would not change.

Transition Independence: Now, we define some notions of *Independence* that will be very important in the conditions of our reduction rules:

Definition 5.7 (Transitions Independence) *Let T_1 and T_2 be two open transitions or two weak open transitions, with*

$$T_1.Post = (x_k \leftarrow e_k)^{k \in K} \qquad and \qquad T_2.Post = (x_{k'} \leftarrow e_{k'})^{k' \in K'}$$

- *T_1, T_2 have Post Independence if T_1's Post would not have any effect on T_2's Post, and vice versa. Formally: $\forall k \in K,\ \forall k' \in K',\ \{x_k\} \cap vars(e_{k'}) = \emptyset \wedge \{x_{k'}\} \cap vars(e_k) = \emptyset$*
- *T_1, T_2 have Pred Independence if T_1's Post would not have any effect on T_2's Pred, and vice versa. Formally: $\forall k \in K,\ \forall k' \in K',\ \{x_k\} \cap vars(T_2.Pred) = \emptyset \wedge \{x_{k'}\} \cap vars(T_1.Pred) = \emptyset$*

- *T_1, T_2 have Hole Action Independence if T_1's Post would not have any effect on T_2's Holes Actions, and vice versa; and moreover the order of T_1 and T_2 would not affect any of their observable Hole action sequences. Formally:* $\forall k \in K,\ \forall k' \in K',\ \{x_k\} \cap vars(T_2.\overline{\beta}) = \emptyset \wedge \{x_{k'}\} \cap vars(T_1.\overline{\beta}) = \emptyset$ *and each Hole in* $(T_1.\overline{\beta})^{\nabla} \cup (T_2.\overline{\beta})^{\nabla}$ *has at most one observable action.*

- *T_1, T_2 have Action Independence if T_1's Post would not have any effect on T_2's Action, and vice versa; and moreover the order of T_1 and T_2 would not affect any of their observable action sequences. Formally:* $\forall k \in K,\ \forall k' \in K',\ \{x_k\} \cap vars(T_2.Action) = \emptyset \wedge \{x_{k'}\} \cap vars(T_1.Action) = \emptyset$ *and each Hole in* $(T_1.\overline{\beta})^{\nabla} \cup (T_2.\overline{\beta})^{\nabla}$ *has at most one observable action.*

- *T_1, T_2 have Transition Independence if they have all Post Independence, Pred Independence, Hole Action Independence, and Action Independence.*

5.2.4 Local FH-Bisimulation

Now we introduce a notion of *Local* FH-Bisimulation, and prove a lemma stating that local FH-Bisimulation preserves Weak FH-Bisimulation. This generic result will later be used to prove that each of our reduction rules preserves weak bisimulation.

Consider that we already have a system described by an open automaton, split into two parts, local and the rest. Now consider another system where only the *'local'* states and transitions are modified w.r.t those of the original OA, but in a way preserving the structure of states/transitions: we use for that a graph (multi-)morphism $\Phi = (\Phi^S, \Phi^T)$ so that Φ^S relates the states between the two systems, and Φ^T relates the transitions coherently, i.e.: if ot is a transition from s_1 to s_2, then $\Phi^T(ot)$ is a transition from $\Phi^S(s_1)$ to $\Phi^S(s_2)$. Moreover, we want the variable sets between the 2 systems to be disjoint; for this, we use a bijective renaming σ_V to create fresh variables. The following Definition 5.8 describes the situation.

In general, the state mapping between these two systems could be arbitrarily 1 to n or n to 1. In this work, to demonstrate our *Reduction* approach, we only use a special case, where we have only two states in the original local-part and only one state in the reduced local part. In our Reduction rules, Φ^S maps two original local states to one combined new state.

Definition 5.8 (Local FH-Bisimulations) *Let* $System = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$*, and split S into a* 'local' *subset $LocalS$ and the rest RS:* $\mathcal{S} = LocalS \uplus RS$*.*

Similarly, we split the transitions into 4 sets, as shown in Figure 11. Local transitions $LocalOT$ are all transitions between states in $LocalS$, including Self-Loops; incoming (resp. outgoing) are transitions from RS to $LocalS$ (resp from $LocalS$ to RS), and RT transitions between states in RS. Let us denote $EffectedT = LocalOT \uplus incoming \uplus outgoing$*. Then* $\mathcal{T} = EffectedT \uplus RT$*.*

Consider $System' = \langle\!\langle J, \mathcal{S}', s_0, V', \mathcal{T}' \rangle\!\rangle$*, with:* $RS' = \sigma_V(RS)$*,* $\mathcal{S}' = LocalS' \uplus RS'$*,* $V' \cap V = \emptyset$*,* $\mathcal{T}' = LocalOT' \uplus incoming' \uplus outgoing' \uplus \sigma_V(RT)$*, and we have a pair of graph multi-morphisms Φ, Ψ such that:*

- *Φ^S is a total surjective function on $LocalS$, formally* $\forall s \in LocalS.\Phi^S(s_1) \subseteq LocalS' \wedge \forall s_2 \in LocalS'.\exists s_1 \in LocalS.s_2 \in \Phi^S(s_1)$*; and the identity (identifying elements as singletons) on the rest of the states in RS.*

- *and symmetrically for Ψ^S, with the correspondence condition:* $\forall s_1 \in \mathcal{S}, \forall s_2 \in \Phi^S(s_1).s_1 \in \Psi^{S'}(s_2)$*.*

Lemma 5.9 (Preservation by local bisimulation) *Let $System$ and $System'$ be Open Automata obeying Definition 5.8, and Φ, Ψ their correspondence morphisms.*

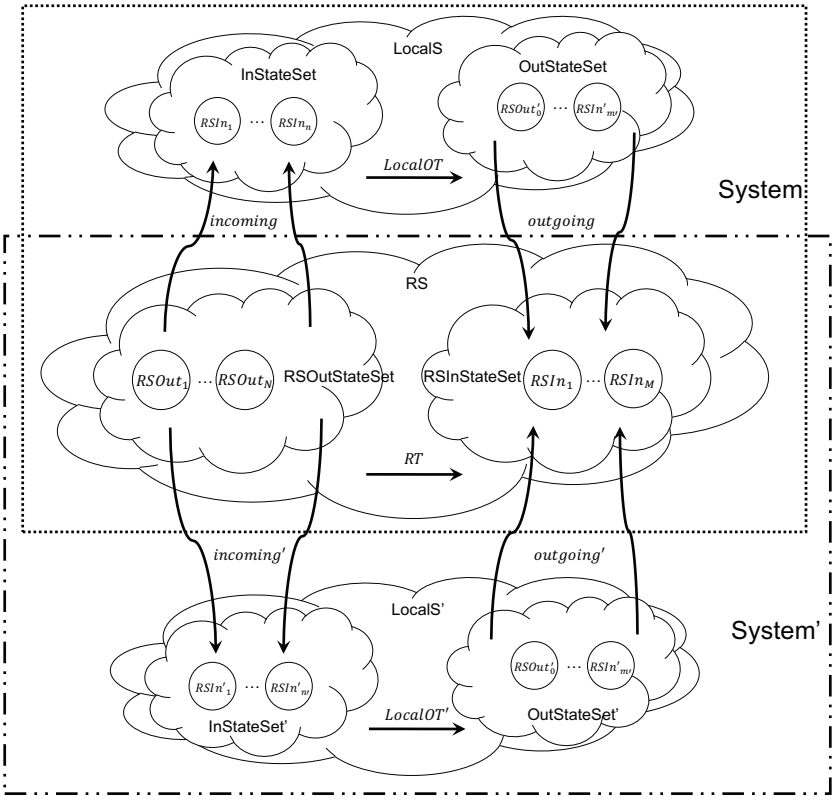


Figure 11: Schema of partial FH-Bisimulation between two systems

Suppose there exists a Tripleset $\mathcal{R} = (s_1 \in \mathcal{S}, s_2 \in \Phi^S(s_1) | Pred_{s_1,s_2})$ *such that:*

- *for each* $ot \in EffectedT$*, there exists an open transition* ot' *with* $ot' \in EffectedT'$ *and* $ot'.sourceState \in \Phi^S(ot.sourceState) \wedge ot'.targetState \in \Phi^S(ot.targetState)$*, and* (ot, ot') *simulate each other in the sense of Definition 5.5,*
- *symmetrically, for each* $ot' \in EffectedT'$*, there exist an open transition* $ot \in EffectedT$ *with* $ot.sourceState \in \Psi^S(ot'.sourceState) \wedge ot.targetState \in \Psi^S(ot'.targetState)$*, and* (ot, ot') *simulate each other,*

then $\mathcal{R}$ *is a Weak FH-Bisimulation relation.*

In other terms, the transformation encoded as Φ *preserves weak bisimulation on the whole system.*

Proof 5.10

Let $System = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$ *and* $System' = \langle\!\langle J, \mathcal{S}', s_0, V', \mathcal{T}' \rangle\!\rangle$ *obeying Definition 5.8,* $\mathcal{WT}$ *and* $\mathcal{WT}'$ *the sets of weak transitions derived from* $\mathcal{T}$ *and* $\mathcal{T}'$*, and TripleSet* $\mathcal{R} = (s_1 \in \mathcal{S}, s_2 \in \Phi^S(s_1) | Pred_{s_1,s_2})$ *verifying the hypotheses of Lemma 5.9.*

According to the definition of weak bisimulation, we need to prove:

- *for any open transition* ot *in* $\mathcal{T}$*:* $ot = \dfrac{\beta_j^{j \in J'}, Pred_{ot}, Post_{ot}}{s_1 \xrightarrow{\alpha} s_1'}$

there exists a set of WOTs $wot_x^{x\in X} \subseteq \mathcal{WT}'$ $wot_x = \dfrac{\gamma_{jx}^{j\in J_x}, Pred_{wot_x}, Post_{wot_x}}{s_2 \overset{\alpha_x}{\Longrightarrow} s'_{2x}}$

with $\forall x, \{j \in J' | \beta_j \neq \tau\} = J_x, (s'_1, s'_{2x} | Pred_{s'_1, s'_{2x}}) \in \mathcal{R}$*; and*

$$Pred_{s_1,s_2} \wedge Pred_{ot} \implies \bigvee_{x\in X} \left(\forall j \in J_x . (\beta_j)^{\nabla} = \gamma_{jx} \wedge Pred_{wot_x} \wedge \alpha = \alpha_x \wedge Pred_{s'_1, s'_{2x}} \{\!\{ Post_{ot} \uplus Post_{wot_x} \}\!\} \right)$$

- *and symmetrically any OT from* s_2 *in* $\mathcal{T}'$ *can be covered by a set of WOTs from* s_1 *in* $\mathcal{WT}$*.*

Remark that we are in a restrictive case of the general conditions for Weak FH-Bisimulation: only a small part of the system is changed between the two systems, and the weak simulation hypothesis constrains the transition correspondence to be one to one, rather than one transition 'covered' by a set on the other side. This will simplify the search for corresponding (weak) transitions from the definition above. Both the goal (above), and the hypotheses (pair of OTs that simulate each other, in the lemma) are symmetric, so we shall only consider one direction in the proof.

The schema of the proof is: for any open transition ot from System, there are 2 main cases: either ot_1 *is in EffectedT, then by hypotheses, we have an* ot_2 *in EffectedT'* *and* (ot_1, ot_2) *simulate each other, then we can construct a weak OT* wot_2 *so that the set* $\{wot_2\}$ *matches* ot_1 *in the sense of Weak FH-Bisimulation. Or* ot_1 *is in RT, then we have an identical (modulo renaming of variables)* ot_2 *in RT', and it's easy to construct* $\{wot_2\}$ *fulfilling the conditions.*

Formally:

Choose any open transition $ot_1 = \dfrac{\overline{\beta_1}, Pred_1, Post_1}{s_1 \xrightarrow{\alpha_1} s'_1} \in \mathcal{T}$*, with* J *its set of active holes,*

(1) *If* $ot_1 \in EffectedT$*, then the lemma hypothesis ensure that:*

there exists an open transition $ot_2 = \dfrac{\overline{\beta_2}, Pred_2, Post_2}{s_2 \xrightarrow{\alpha_2} s'_2} \in \mathcal{T}'$*, with the same* J*, so that* (ot_1, ot_2) *simulate each other.*

Let $(s_1, s_2 | Pred_{s_1,s_2})$ *and* $(s'_1, s'_2 | Pred_{s'_1,s'_2})$ *be the corresponding Triples in* $\mathcal{R}$*.* (ot_1, ot_2) *'simulate each other' implies that the set of active holes of* ot_2 *is also* J*, and that there exists a WOT:*

$$wot_2 = \frac{\overline{\gamma_2}, Pred_2, Post_2}{s_2 \overset{\alpha_2}{\Longrightarrow} s'_2}, \quad with \forall j \in J, \gamma_{2j} = (\beta_{2j})^{\nabla}$$

such that the following formula is true:

$$Pred_{s_1,s_2} \wedge Pred_1 \implies \forall j \in J . (\beta_{1j})^{\nabla} = \gamma_{2j} \wedge Pred_2 \wedge \alpha_1 = \alpha_2 \wedge Pred_{s'_1,s'_2} \{\!\{ Post_1 \uplus Post_2 \}\!\} \quad (6)$$

So we can construct a set of weak transitions reduced to a singleton $\{wot_2\}$ *and check the conditions of the Weak FH-Bisimulation:*

- *The set of active holes is the same.*
- *The target states are equivalent:* $(s'_1, s'_2 | Pred_{s'_1,s'_2}) \in \mathcal{R}$

- *and the coverage condition from Equation (5.10) becomes:*

$$Pred_{s_1,s_2} \wedge Pred_1 \implies \forall j \in J.(\beta_j)^{\nabla} = \gamma_j \wedge \ Pred_2 \wedge \alpha = \alpha_x \wedge Pred_{s'_1,s'_2}\{\!\{Post_1 \uplus Post_2\}\!\}$$

That is exactly Formula (6).

(2) *Otherwise $ot_1 \in RT$, then the situation is simpler. According to Definition 5.8, we have $RS' = \sigma_V(RS)$.*

Let's construct:

$$ot2 = \frac{\overline{\beta_1}\{\!\{\sigma_V\}\!\}, Pred_1\{\!\{\sigma_V\}\!\}, Post_1\{\!\{\sigma_V\}\!\}}{\sigma_V(s1) \xrightarrow{\alpha_1\{\!\{\sigma_V\}\!\}} \sigma_V(s1')} \in \mathcal{T}'$$

and using WT1, turn it to a weak OT:

$$wot2 = \frac{\overline{\gamma}, Pred_1\{\!\{\sigma_V\}\!\}, Post_1\{\!\{\sigma_V\}\!\}}{\sigma_V(s1) \stackrel{\alpha_1\{\!\{\sigma_V\}\!\}}{\Longrightarrow} \sigma_V(s1')} \in \mathcal{WT}'$$

Let $(s_1, \sigma_V(s_1)|Pred_{s_1,\sigma_V(s_1)})$ and $(s'_1, \sigma_V(s'_1)|Pred_{s'_1,\sigma_V(s'_1)})$ be the corresponding Triples in $\mathcal{R}$.

Now it is easy to check the conditions of the Weak FH-Bisimulation for the singleton $\{wot_2\}$:

- *The set of active holes is the same.*
- *The target states are equivalent: $(s'_1, \sigma(s'_1)|Pred_{s'_1,\sigma(s'_1)}) \in \mathcal{R}$*
- *and the coverage condition from Equation (5.10) becomes:*

$$\begin{array}{l} Pred_{s_1,\sigma(s_1)} \wedge Pred_1 \implies \\ \quad \forall j \in J.(\beta_{1j})^{\nabla} = \gamma_j \wedge Pred_1\{\!\{\sigma_V\}\!\} \wedge \alpha_1 = \alpha_1\{\!\{\sigma_V\}\!\} \wedge Pred_{s'_1,\sigma(s'_1)}\{\!\{Post_1 \uplus Post_1\{\!\{\sigma_V\}\!\}\}\!\} \end{array} \tag{7}$$

For a state s in RS, let us denote $Pred_{s,\sigma_V(s)} = \Big\{ \bigwedge_{v \in vars(s)} v = \sigma_V(v) \Big\}$. We have :

$$\begin{array}{l} Pred_{s_1,\sigma_V(s_1)} \implies \forall j \in J.\beta_{1j} = \beta_{1j}\{\!\{\sigma_V\}\!\} \\ Pred_{s_1,\sigma_V(s_1)} \implies \alpha_1 = \alpha_1\{\!\{\sigma_V\}\!\} \\ Pred_{s_1,\sigma_V(s_1)} \implies Post_1 = Post_1\{\!\{\sigma_V\}\!\} \\ Pred_{s_1,\sigma_V(s_1)} \wedge Pred_1 \implies Pred_1\{\!\{\sigma_V\}\!\} \end{array}$$

Then formula 7 is tautology. The proof of the other direction is similar, so this ends the proof of Lemma 5.9.

5.3 Three reduction rules

Given a open automaton $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$, we have identified three patterns for reduction. Any application of one of the rules decreases the number of states. We treat the result open automaton as an independent $A' = \langle\!\langle J, \mathcal{S}', s'_0, V\{\!\{\sigma_V\}\!\}, \mathcal{T}' \rangle\!\rangle$, in order to match the definition of (Weak) Bisimulation. Here, substitution $\sigma_V = \bigcup_{v \in V} (v \leftarrow v')$ has the same meaning as the one from Definition 5.1 which gives a new name to each variable.

5.4 Three reduction rules

Given a open automaton $A = \langle\!\langle J, \mathcal{S}, s_0, V, \mathcal{T} \rangle\!\rangle$, we have identified three patterns for reduction. Any application of one of the rules decreases the number of states. We treat the result open automaton as an independent $A' = \langle\!\langle J, \mathcal{S}', s_0', V\{\!\{\sigma_V\}\!\}, \mathcal{T}' \rangle\!\rangle$, in order to match the definition of (Weak) Bisimulation. Here, substitution $\sigma_V = \bigcup_{v \in V} (v \leftarrow v')$ has the same meaning as the one from Definition 5.1 which gives a new name to each variable.

5.4.1 Rule of tau-Merging

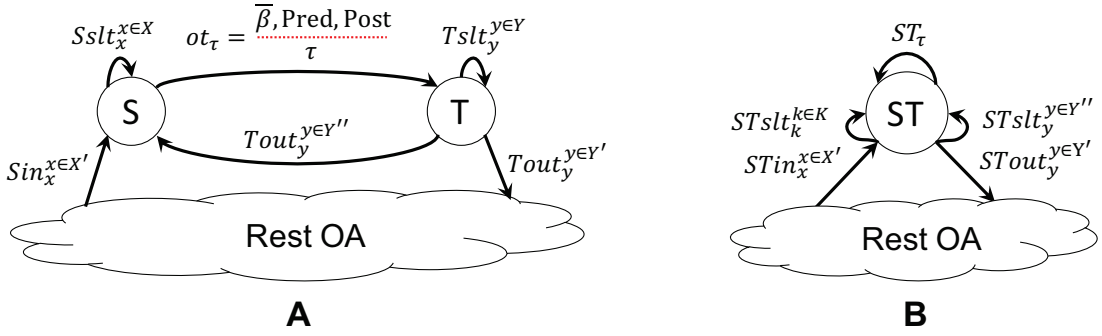


Figure 12: τ-Merging Pattern, **(A)** is the pattern before τ-Merging, **(B)** is the pattern after τ-Merging.

Here ot_τ is an OT with a non-observable action, represented here by τ, and very simple premises, namely: $(\overline{\beta} = \overline{\tau} \ \vee \ \overline{\beta} = \emptyset) \wedge Pred = True \wedge Post = \{\}$.

We have states $S, T \in \mathcal{S}$ and ot_τ is the only transition which starts from S and ends in T, and there isn't any other transition starting from S or ending in T except Self-Loop transitions, as shown in Figure 12A.

Under any of following conditions, we can merge ot_τ with $Tout_y^{y\in Y'}$, $Tout_y^{y\in Y''}$ and merge states S and T.

- T and S don't have Self-Loop transitions, $Y = X = \emptyset$.
- T doesn't have Self-Loop transitions but S has. $Y = \emptyset \wedge X \neq \emptyset$.
- S doesn't have Self-Loop transition but T has. $X = \emptyset \wedge Y \neq \emptyset$.
- T and S both have Self-Loop transitions, and $Sslt_x^{x\in X} \overset{WS}{\Longleftrightarrow} Tslt_y^{y\in Y}$.

The procedure of merging S and T goes as follows: create a new state, say ST, then modify the target state of all transitions coming in S to ST, and change the source state of all transitions come out of T to ST. The result shows in Figure 12B.

We should pay attention to the two sets of Self-Loop transitions sets, $STslt_k^{k\in K}$, $STslt_y^{y\in Y'}$ and $STslt_y^{y\in Y''}$. If X and Y are not empty sets and $Sslt_x^{x\in X} \overset{WS}{\Longleftrightarrow} Tslt_y^{y\in Y}$, then we only need to

'copy' one of transition sets (X or Y) to the new state ST. In other words, $STslt_k^{k\in K}$ comes from $Sslt_x^{x\in X}$ or $Tslt_y^{y\in Y}$. $STslt_y^{y\in Y'}$ comes from the result of merging ot_τ and $Tout_y^{y\in Y'}$. $STslt_y^{y\in Y''}$ comes from the result of merging ot_τ and $Tout_y^{y\in Y''}$. After merging states S and T, it becomes a new set of Self-Loop transitions. Last, ST_τ self-loop is required as a reflection of ot_τ, to match the requirements of local-Bisimulation.

Table 3: Relation Triples for Tau Merging

Original OA	Result OA	Predicate
S	ST	$\bigwedge_{v\in V} v = v'$
T	ST	$\bigwedge_{v\in V} v = v'$
$\forall s \in \mathcal{S}/\{S,T\}$	$\sigma_V(s)$	$\bigwedge_{v\in V} v = v'$

Theorem 5.11 *Applying a τ-Merging on an OA, the result has weak bisimulation relation with the original OA with TripleSet defined by Table 3.*

Here is the proof of the above theorem.

Proof 5.12 *Using lemma 5.9, we simply need to prove here that each transition in the upper part of the automaton in 12A has a corresponding OT in 12B, and that they simulate each other.*

With the notion of Similar in section 5.2.3, and the pre-conditions, it is easy to see that transitions in X are Similar with the transitions in Y (X and Y can be empty set).

The procedure of τ-Merging only changes the source state and target state of transitions. And merging ot_τ with $Tout_y^{y\in Y'}$ or $Tout_y^{y\in Y''}$ will not change any semantic meaning (except source state and target state). So with Pred $\left[\bigwedge_{v\in V} v = v'\right]$ in Triples, $Sin_x^{x\in X'}$ and corresponding $STin_x^{x\in X'}$ simulate each other; $Sslt_x^{x\in X}$ and corresponding $STslt_k^{k\in K}$ simulate each other; $Tslt_y^{y\in Y}$ and corresponding $STslt_k^{k\in K}$ simulate each other; $Tout_y^{y\in Y'}$ and corresponding $STout_y^{y\in Y'}$ simulate each other; $Tout_y^{y\in Y''}$ and corresponding $STslt_y^{y\in Y''}$ simulate each other.

ot_τ and ST_τ simulate each other:

The only thing left is that we need to find a WOT from result OA that simulates the transition ot_τ. It can be simulated by a self-loop τ transition on state ST; let us name it ST_τ.

$$ST_\tau = \frac{\overline{\beta}, True, \emptyset}{ST \xrightarrow{\tau} ST}$$

In ot_τ, we have $\overline{\beta} = \overline{\tau} \vee \overline{\beta} = \emptyset$, $Pred = True$, $Post = \emptyset$ and the resulting action is non-observable.

The proof obligation is:

$$Pred_{S,ST} \wedge True$$

$$\Longrightarrow \left[\bigwedge_{j\in J} \tau = \tau\right] \wedge True \wedge \tau = \tau \wedge Pred_{T,ST}$$

where J is empty when $\overline{\beta} = \emptyset$.

With Triple Predicates $Pred_{S,ST} = Pred_{T,ST}$, the proof obligation is a tautology, so (ot_τ, ST_τ) simulate each other, we can apply Lemma 5.9, and conclude that τ-Merging preserves Weak FH-Bisimulation.

Remark that in case we want to guarantee non-observability of τ actions, we would have both kind of ot transitions in the original OA, so the rule would have to be extended allow τ-Merging.

5.4.2 Forward Merging

Figure 13 shows the schema for the Forward Merging rule. Here *ot* is the only input transition of T besides Self-Loop transitions in T, shown in the pattern on the left. In other words, for all possible transition sequences which pass S and T, *ot* is the only previous transition of any transition that comes out of T.

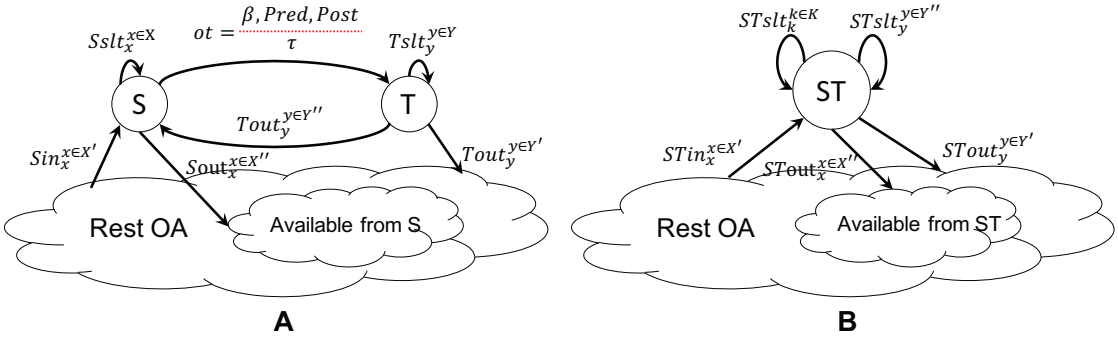


Figure 13: Forward Merging Pattern, **(A)** is the pattern before Forward Merging, **(B)** is the pattern after Forward Merging.

When *ot* has a non-observable action, and $\overline{\beta} = \overline{\tau} \vee \overline{\beta} = \emptyset$, we can merge *ot* with each transition from T ($Tout_y^{y\in Y'}$ and $Tout_y^{y\in Y''}$) and merge states S and T under any one of the following conditions:

- T and S have no Self-Loop transitions, $Y = \emptyset \wedge X = \emptyset$.
- T has no Self-Loop transitions but S has. $Y = \emptyset \wedge X \neq \emptyset$.
- S has no Self-Loop transition but T has. $X = \emptyset \wedge Y \neq \emptyset$. Then it should match the following conditions:
  - *ot* has *Transition Independence* with any transition in $Tslt_y^{y\in Y}$.
- T and S both have Self-Loop transitions, then it should match the following conditions:
  - $Sslt_x^{x\in X} \overset{WS}{\Longleftrightarrow} Tslt_y^{y\in Y}$.
  - *ot* has *Transition Independence* with any transition in $Sslt_x^{x\in X}$ and $Tslt_y^{y\in Y}$.

The result shows in Figure 13B:
$STout_y^{y\in Y'}$ is the result of applying **MT1** on *ot* and each transition in $Tout_y^{y\in Y'}$.
$STslt_y^{y\in Y''}$ is the result of applying rule **MT1** on *ot* and each transition in $Tout_y^{y\in Y''}$.

$STslt_k^{k\in K}$ is the union of $Sslt_x^{x\in X}$ and $Tslt_y^{y\in Y}$. If $X \neq \emptyset \wedge Y \neq \emptyset$, then we have $Sslt_x^{x\in X}$ and $Tslt_y^{y\in Y}$. So, it's OK to only keep any one of $Sslt_x^{x\in X}$ and $Tslt_y^{y\in Y}$.

With $\left[\bigwedge_{v\in V} v = v'\right]$, we will have:

$STin_x^{x\in X'}$ is *Similar* to $Sin_x^{x\in X'}$; $STout_x^{x\in X''}$ is *Similar* to $Sout_x^{x\in X''}$.

5.4.3 Backward Merging

Here *ot* is the only outgoing transition of S besides Self-Loop transitions in S, as illustrated in Figure 14A. In other words, for all possible transition sequences which pass S and T, *ot* is the only next transition of any transition that comes in S.

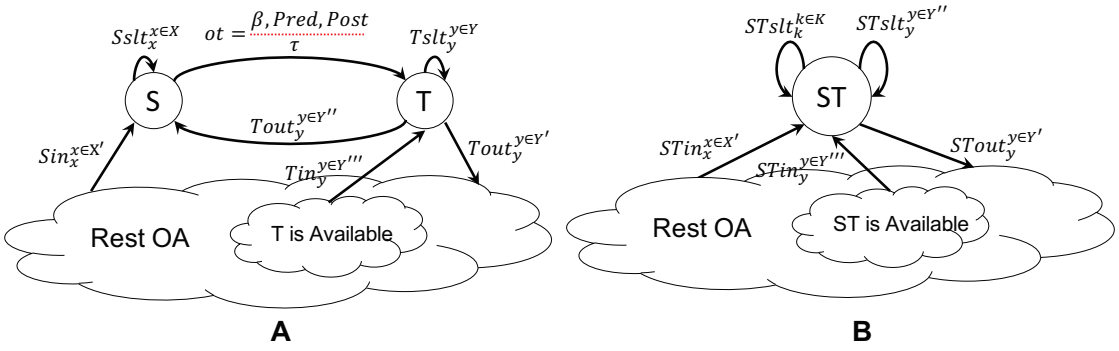


Figure 14: Backward Merging Pattern, **(A)** is the pattern before Backward Merging, **(B)** is the pattern after Backward Merging.

When *ot* has a non-observable action and $\overline{\beta} = \overline{\tau} \ \vee \ \overline{\beta} = \emptyset$, we can merge *ot* with each input transition of S, $Sin_x^{x\in X'}$, $Tout_y^{y\in Y''}$, and combine state S and T under any one of the following conditions:

- T and S don't have Self-Loop transition, $Y = \emptyset \wedge X = \emptyset$.
- T doesn't have Self-Loop transition but S has. $X \neq \emptyset \wedge Y = \emptyset$. Then it should match the following conditions:
  - *ot* has *Transition Independence* with any transition in $Sslt_x^{x\in X}$.
  - Any transition in $Tout_y^{y\in Y''}$ has *Transition Independence* with any transition in $Sslt_x^{x\in X}$.
- S doesn't have Self-Loop transition but T has. $X = \emptyset \wedge Y \neq \emptyset$.
- S and T both have Self-Loop transitions, then it should match the following conditions:
  - $Sslt_x^{x\in X} \overset{WS}{\Longleftrightarrow} Tslt_y^{y\in Y}$.
  - *ot* has *Transition Independence* with any transition in $Sslt_x^{x\in X}$ and $Tslt_y^{y\in Y}$.
  - Any transition in $Tout_y^{y\in Y''}$ has *Transition Independence* with any transition in $Sslt_x^{x\in X}$ and $Tslt_y^{y\in Y}$.

The result shows in Figure 14B:

$STin_x^{x\in X'}$ is the result of applying **MT2** on *ot* and each transitions in $Sin_x^{x\in X'}$.

$STslt_y^{y\in Y''}$ is the result of applying rule **MT2** on *ot* and each transition in $Tout_y^{y\in Y''}$.

$STslt_k^{k\in K}$ is the union of $Sslt_x^{x\in X}$ and $Tslt_y^{y\in Y}$. If $X \neq \emptyset \wedge Y \neq \emptyset$, then we have $Sslt_x^{x\in X}$ and $Tslt_y^{y\in Y}$. So, it's OK to only keep any one of $Sslt_x^{x\in X}$ and $Tslt_y^{y\in Y}$.

With $\Big[\bigwedge_{v\in V} v = v'\Big]$, we will have:

$STout_y^{y\in Y'}$ is *Similar* to $Sout_y^{y\in Y'}$; $STin_y^{y\in Y'''}$ is *Similar* to $Tin_y^{y\in Y'''}$.

The three principles and Merging rules construct the Reduction algorithm. It is called Reduction because the minimality cannot be guaranteed. But it is effective in our use-case below.

5.5 Reduce the Running Example

Let us apply these three rules for the Implementation OA of our example in Figure 15A (we don't give fresh names to variables for readability).

Table 4: Relation Triples for IOA_2 and IOA_1

IOA_2	IOA_1	Predicate
S1	000	True
S1	202	True
S4	100	`s_msg' = s_msg` $\wedge$ `s_ec' = s_ec`
S4	210	`s_msg' = m_msg` $\wedge$ `s_ec' = m_ec`
S4	220	`s_msg' = s_msg` $\wedge$ `s_ec' = s_ec`
S4	201	`s_msg' = r_msg` $\wedge$ `s_ec' = r_ec`

We have ot_τ between two pairs of states: SI_τ between (202,000), and SI_5 between (210,220). The states in each pair have identical Self-Loop transitions. We use τ-Merging in both cases getting new state S_1 from state pairs (000,202) and new state S_2 from (220,210). The rest remains the same. We get the 1st result of minimization, Figure 15B.

For SI_7, it is the only outgoing transition from state S_2 and ending in state 201. It matches the conditions for *Forward Merging*. Then we can apply this rule, which merges state S_2 and 201, and returns a new state S_3. It is the 2nd result of minimization, Figure 15C.

Table 5: Relation Triples for IOA_2 and *SOA*

IOA_2	*SOA*	Predicate
S1	b0	True
S1	b0	True
S4	b1	`s_msg' = b_msg` $\wedge$ `s_ec' = b_ec`
S4	b1	`s_msg' = b_msg` $\wedge$ `s_ec' = b_ec`
S4	b1	`s_msg' = b_msg` $\wedge$ `s_ec' = b_ec`
S4	b1	`s_msg' = b_msg` $\wedge$ `s_ec' = b_ec`

We continue the processing from Figure 15C. We find that, for SI_4, it is the only transition that comes in S_3, apart from Self-Loop transitions. Moreover, the Self-Loop transitions are the same from state 100 and S_3. We can apply rule *Forward Merging* to merge the state pair (100,S_3), getting a new state S_4 and the third result of minimization. Then we cannot find any case that satisfies any one of our reduction rules. It is the final result.

Let us call IOA_1 the original Implementation OA, IOA_2 it's final reduced OA (Figure 15D), and *SOA* the specification OA. Even if we have not proven that forward merging preserves Weak FH-Bisimulation, we can check that indeed IOA_1 and IOA_2 are weak bisimilar, using the TripleSet in table 4. In these triples, the primed variables are the fresh variables obtained by the successive renamings.

Moreover, the reduced OA is much easier to compare with the Specification OA from Figure 2 because we have now two states matching those of the Specification OA.

Variables in s_msg' and s_ec' in IOA_2 correspond to the Specification OA variables b_msg and b_ec, and we can prove their equivalence using the TripleSets in Table 5.

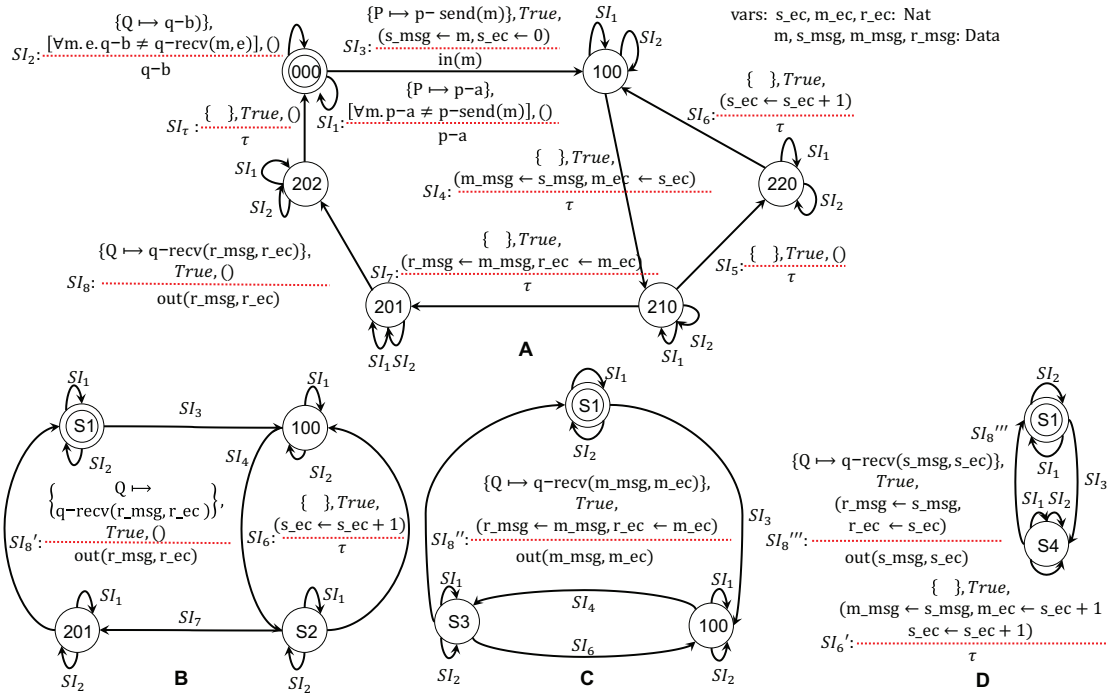


Figure 15: Minimization of the Implementation OA **(A)** is the original OA, **(B)** is the 1st result of Reduction, **(C)** is the 2nd result of Rection, **(D)** is the 3rd result of Reduction.

5.6 Discussion

OSM is very "efficient" in minimizing OA's size and hides the detailed structural information, which increases the difficulties of reading and understanding. So, it is not very useful in practices. That is why we propose the 'Reduction' method, which reduces the size while preserving the structure of the automaton.

The main idea of 'Reduction' is based on merging non-observable transitions with others and combining the neighboring states. And naturally, we want the 'Reduction' to preserve weak bisimulation.

We can see that the Reduction Rules have two significant components, merging transitions

and combining states. Merging transitions follows the **WT3** in the construction of WOTs, but it is split into two parts by the merging directions of non-observable action, forward and backward. The rules of states combination are based on graph patterns. The τ-Merging removes the meaningless τ transitions in the system. Forward or Backward Merging reduces the number of states by merging transitions with its neighbors. After each step of the above reductions, the result OA has weak bisimulation with the original OA as ensured using the partial FH-Bisimulation Lemma 5.9.

From our Implementation OA of our running example, all self-loop transition sets, on all states, simulate each other. Furthermore, the self-loop transitions have *Transition Independence* with all the other transitions except SI_3 and SI_8. These just fit the requirements of the three Reduction rules. In other words, we do not have too many dependence obstacles but finding the graph patterns.

On more complex examples, where predicates and posts would have complex interplay, our rules will certainly not apply, because of the non-interfe-rence (*Transition Independence*) condition. More complex rules may certainly be crafted, but it may be tricky to find conditions allowing for an automatic algorithmic usage of the rules, while still having a formal proof that such rules respect the local FH-Bisimulation lemma. This is already the case here, as our forward-merging and backward merging rules do not fulfill the requirements to apply the lemma. We need to change them slightly (both the rules and the lemma) to get a preservation proof; this is work in progress. Whenever a rule is used that has not been formally proved to preserve weak-FH-Bisimulation, it is necessary to exhibit some fitting TripleSet, and apply the (weak) bisimulation checking algorithm between the original and the reduced OA.

6 Related Work

There are many works on Bisimulation and Minimization, but not too many on Weak Bisimulation Equivalence and Minimization with complex automata model like ours which is open, symbolic, data-aware, and contains loops, assignments (Post), guards (Pred). In this section, we give a brief overview of related work on Weak Bisimulation and Minimization.

Open and Compositional systems: Johnson et al. [13, 12] investigate several methods to verify component-based software systems. The authors use SMT-solver for independent subproblems on independent sets of variables from the decoupling of verification problems. Our theory and implementation have also encoded into SMT-solver (Z3), but with more structured. Our pNets model and theory are quite different from them.

Bisimulation: As mentioned in *introduce*, H.M. Lin's works [7] [11] [15] inspire us. He created symbolic transition graphs (STG) and based that he developed the theories about early/late/strong/weak symbolic bisimulation. He addressed value-passing calculi and used it in symbolic transition graphs with assignments (STGA). Lin gave out a semantic theory survey for value-passing processes, focusing on bisimulation equivalences with early or late semantic. Van [22] proposed a kind of equivalence, branching bisimulation, which is stronger than weak bisimulation but weaker than strong bisimulation. It preserves the branching structure of the process.

However, our model is more scalable because of the 'Open' property, and less restrictive on the data type for the powerful SMT engine. Furthermore, this paper is focusing on the realization of Weak Bisimulation on OA.

There is another kind of "Symbolic Bisimulation" research domain, namely BDD-like techniques for modeling and computing finite-state bisimulations. It is not related to our topic.

Minimization and Reduction: When searching 'Minimization Automata', there are lots

of works on traditional automata. Loris [3] proved the basic properties of minimality in the symbolic setting, and lift classical minimization algorithms (Huffman-Moore's and Hopcroft's algorithms) to finite symbolic automata. Richard [17] presents an efficient algorithm based on transition pruning techniques to reduce the size of nondeterministic Buchi word automata while retaining their language. However, as mentioned before, our model is open and data-aware with assignments.

Radu's solutions [16] inspire us a lot where the idea of τ-**Merging** comes from *Tau-compression* and the idea of **Forward Merging** and **Backward Merging** comes from *Tau-confluence* in that paper but with concerns on data assignments and guards, precisely all kinds of *Independence*. And our solutions is more pattern based.

7 Conclusion

pNets is a formalism adapted to the representation of the behavior of a parallel or distributed systems. Open pNets are pNets with unspecified sub-systems or processes known as 'Holes'. Open pNets are the hierarchical composition of automata with Holes and parameters which can be transformed to open automata (a kind of LTS with data and assignments) by semantics but without hierarchy. The previous works defined a strong bisimulation and a weak bisimulation, which are based on so-called FH-Bisimulation.

This article explores two ways to implement weak bisimulation: **(1)** construct the complete weak open automata (WOA) and weak open transitions (WOT); **(2)** directly build expected WOT on-the-fly when searching. We propose several definitions, meta-Variable, meta-WOT, meta-WOA, and meta FH-Bisimulation, for the first approach. However, we found it is not easy to generate meta-Variables from the infinity Self-Loop transition sequences.

We then analyze the second approach and find out it is easier for understanding and implementing it. Two search styles for 'expected WOTs' are used, DFS and BFS, both with searching bound. We implement it into project VerCors, and we provide a detailed technical document in the appendix.

Besides that, we also explore minimization and reduction. One State Minimization can be implemented to any OA and returns the result OA with only one state, which has strong bisimulation relation with the original one. We propose three 'reduction' principles, graph rewriting rules with conditions, for OA reduction, and give a (partial) proof that the reduction preserves Weak FH-Bisimulation.

7.1 Future Work

Of course there is still some work for the future. We will complete more examples of OA and weak bisimulation. In the current state of the work, only one of our reduction rules is formally proven to preserve weak bisimulation. We are working on extending this result, in order to be able to use reduction without being required to prove weak bisimulation of the result process on each use-case. We plan to validate the Reduction approach, whether it is useful for realistic (from the industry) and complex (structures of the system) use-cases, where compositionality is especially important. Another, more immediate work would be to deal with big examples where reduction can be combined with hierarchical construction of the OAs, and understand if this allows to deal with bigger applications.

The use of Meta-variables gives excellent scalability for our model to handle parameters explicitly in particular cases. It provides a good solution to manage Looping transitions. But meta-Variables could also be used by users at design time for finite structures encoded with some data parameters, very often found in software architecture designs, like arrays, matrices,

rings, etc. Using meta-Variables this way would require a significant extension of our framework, both at the level of the formalism: extending the Open Automaton structure, and the semantic constructions generating them from high-level formalisms and languages; then extending the strong FH-Bisimulation definition, and the equivalence checking algorithms. In fact, checking (strong or weak) StrFH-Bisimulation between two open automata with meta-Variables may be quite different in practice, because if we have a meta-Variable in the source OT of a *simulation* comparison, rather than a specific valuation of the variable, then we cannot used our on-the-fly approach: the bounded search cannot find a transition corresponding to all possible values of this variable. Maybe this would require using some theorem-proving approaches rather than our finite automatic search method.

But the perspectives are very interesting, as this could lead to a whole field of applications, as mentioned above, that is not easily addressed by compositional methods today. As a simple example, consider transport protocols: our simple example in this paper was on purpose very small and unrealistic. But protocols effectively used on the web (transport, broadcast, security protocols) are of course complex, and must often be modeled as systems with a parametric topology, that could be encoded as meta-OAs. Then full proofs of correctness using the open automaton approach seems feasible.

References

[1] Rabéa Ameur-Boulifa, Ludovic Henrio, and Eric Madelaine. *Compositional equivalences based on Open pNets*. https://arxiv.org/abs/2007.10770. 2020. arXiv: 2007.10770 [cs.LO].

[2] Jos CM Baeten and W Peter Weijland. "Process Algebra, volume 18 of". In: *Cambridge tracts in theoretical computer science* (1990), pp. 141–147.

[3] Loris D'Antoni and Margus Veanes. "Minimization of symbolic automata". In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 2014, pp. 541–553. DOI: 10.1145/2535838.2535849. URL: https://doi.org/10.1145/2535838.2535849.

[4] Rocco De Nicola. "Behavioral Equivalences". In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Boston, MA: Springer US, 2011, pp. 120–127. ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_517. URL: https://doi.org/10.1007/978-0-387-09766-4_517.

[5] R. De Simone. "Higher-level Synchronising Devices in MEIJE-SCCS". In: *Theoretical Computer Science* 37 (1985), pp. 245–267. DOI: 10.1016/0304-3975(85)90093-3. URL: https://doi.org/10.1016/0304-3975(85)90093-3.

[6] Matthew Hennessy and Huimin Lin. "Symbolic Bisimulations". In: *Theoretical Computer Science* 138.2 (1995), pp. 353–389. DOI: 10.1016/0304-3975(94)00172-F. URL: http://dx.doi.org/10.1016/0304-3975(94)00172-F.

[7] Matthew Hennessy and Huimin Lin. "Symbolic bisimulations". In: *Theoretical Computer Science* 138.2 (1995), pp. 353–389. DOI: 10.1016/0304-3975(94)00172-F. URL: https://doi.org/10.1016/0304-3975(94)00172-F.

[8] Matthew Hennessy and Julian Rathke. "Bisimulations for a Calculus of Broadcasting Systems". In: *Theoretical Computer Science* 200.1-2 (1998), pp. 225–260. DOI: 10.1016/S0304-3975(97)00261-2. URL: http://dx.doi.org/10.1016/S0304-3975(97)00261-2.

[9] Ludovic Henrio, Eric Madelaine, and Min Zhang. "A Theory for the Composition of Concurrent Processes". In: *36th International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE)*. Ed. by Elvira Albert and Ivan Lanese. Vol. LNCS-9688. Formal Techniques for Distributed Objects, Components, and Systems. Heraklion, Greece, June 2016, pp. 175–194. URL: https://hal.inria.fr/hal-01432917.

[10] Zechen Hou and Eric Madelaine. "Symbolic Bisimulation for Open and Parameterized Systems". In: *PEPM 2020 - ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*. New-Orleans, United States, Jan. 2020. DOI: 10.1145/3372884.3373161. URL: https://hal.inria.fr/hal-02406098.

[11] Anna Ingolfsdottir and Huimin Lin. "A symbolic approach to value-passing processes". In: *Handbook of Process Algebra*. Elsevier, 2001, pp. 427–478. DOI: 10.1016/B978-044482830-9/50025-4. URL: https://doi.org/10.1016/B978-044482830-9/50025-4.

[12] Kenneth Johnson and Radu Calinescu. "Efficient re-resolution of SMT specifications for evolving software architectures". In: *Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures*. 2014, pp. 93–102. DOI: 10.1145/2602576.2602578. URL: https://doi.org/10.1145/2602576.2602578.

[13] Kenneth Johnson, Radu Calinescu, and Shinji Kikuchi. "An incremental verification framework for component-based software systems". In: *Proceedings of the 16th International ACM Sigsoft symposium on Component-based software engineering*. 2013, pp. 33–42. DOI: 10.1145/2465449.2465456. URL: https://doi.org/10.1145/2465449.2465456.

[14] Kim G. Larsen. "A Context Dependent Equivalence Between Processes". In: *Theoretical Computer Science* 49 (1987), pp. 184–215. DOI: 10.1016/0304-3975(87)90007-7. URL: https://doi.org/10.1016/0304-3975(87)90007-7.

[15] H.M. Lin. "Symbolic Transition Graph with Assignment". In: *Concur'96*. Ed. by Ugo Montanari and Vladimiro Sassone. Vol. 1119. LNCS. Springer, Heidelberg, 1996, pp. 50–65. DOI: 10.1007/3-540-61604-7_47. URL: https://doi.org/10.1007/3-540-61604-7_47.

[16] Radu Mateescu. "On-the-fly state space reductions for weak equivalences". In: *Proceedings of the 10th international workshop on Formal methods for industrial critical systems*. 2005, pp. 80–89. DOI: 10.1145/1081180.1081191. URL: https://doi.org/10.1145/1081180.1081191.

[17] Richard Mayr and Lorenzo Clemente. "Advanced automata minimization". In: *ACM SIGPLAN Notices* 48.1 (2013), pp. 63–74. DOI: 10.1145/2480359.2429079. URL: https://doi.org/10.1145/2480359.2429079.

[18] R. Milner. *A Calculus of Communicating Systems*. Berlin, Heidelberg: Springer-Verlag, 1982. ISBN: 0387102353. URL: https://doi.org/10.1007/3-540-10235-3.

[19] R. Milner. *Communication and Concurrency*. Int. Series in Computer Science. SU Fisher Research 511/24. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[20] Xudong Qin et al. "Using SMT engine to generate Symbolic Automata". In: *18th International Workshop on Automated Verification of Critical Systems (AVOCS 2018)*. Electronic Communications of the EASST, 2018. DOI: 10.14279/tuj.eceasst.76.1103. URL: http://dx.doi.org/10.14279/tuj.eceasst.76.1103.

[21] Rob J Van Glabbeek. "The linear time-branching time spectrum I. The semantics of concrete, sequential processes". In: *Handbook of process algebra*. Elsevier, 2001, pp. 3–99. DOI: 10.1016/B978-044482830-9/50019-9. URL: https://doi.org/10.1016/B978-044482830-9/50019-9.

[22] Rob J Van Glabbeek and W Peter Weijland. "Branching time and abstraction in bisimulation semantics". In: *Journal of the ACM (JACM)* 43.3 (1996), pp. 555–600. DOI: 10.1145/233551.233556. URL: https://doi.org/10.1145/233551.233556.

A Details of Implementation of the Example

Details of the meta-WOTs is listed here (Figure 5):

In the first 3 weak transitions, S denotes the set of all global states.

$$W_\tau = \frac{\{\}, True, ()}{S \overset{\tau}{\Rightarrow} S}$$

$$WI_1 = \frac{\{\texttt{P}\mapsto\texttt{p-a}\}, [\forall \texttt{m}.\texttt{p-a} \neq \texttt{p-send(m)}], ()}{S \overset{\texttt{p-a}}{\Longrightarrow} S}$$

$$WI_2 = \frac{\{\texttt{Q}\mapsto\texttt{q-b}\}, [\forall \texttt{m,ec}.\texttt{q-b} \neq \texttt{q-recv(m,ec)}], ()}{S \overset{\texttt{q-b}}{\Longrightarrow} S}$$

All following transitions are parametrized by an arbitrary non-negative integer $n \in \mathcal{N}$.

$$WI_3(n) = \frac{\{\texttt{P}\mapsto\texttt{p-send(m)}\}, True, (\texttt{s\_msg} \leftarrow \texttt{m}, \texttt{s\_ec} \leftarrow n)}{\{000, 202\} \overset{\texttt{in(m)}}{\Longrightarrow} 100}$$

$$WI_{3a}(n) = \frac{\{\texttt{P}\mapsto\texttt{p-send(m)}\}, True, (\texttt{m\_msg} \leftarrow \texttt{m}, \texttt{m\_ec} \leftarrow n, \texttt{s\_ec} \leftarrow n)}{\{000, 202\} \overset{\texttt{in(m)}}{\Longrightarrow} 210}$$

$$WI_{3b}(n) = \frac{\{\texttt{P}\mapsto\texttt{p-send(m)}\}, True, (\texttt{s\_ec} \leftarrow n)}{\{000, 202\} \overset{\texttt{in(m)}}{\Longrightarrow} 220}$$

$$WI_{3c}(n) = \frac{\{\texttt{P}\mapsto\texttt{p-send(m)}\}, True, (\texttt{r\_msg} \leftarrow \texttt{m}, \texttt{r\_ec} \leftarrow n)}{\{000, 202\} \overset{\texttt{in(m)}}{\Longrightarrow} 201}$$

$$WI_4(n) = \frac{\{\}, True, (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec} + n, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \overset{\tau}{\Rightarrow} 210}$$

$$WI_{4a}(n) = \frac{\{\}, True, (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \overset{\tau}{\Rightarrow} 220}$$

$$WI_5(n) = \frac{\{\}, True, (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{210 \overset{\tau}{\Rightarrow} 220}$$

$$WI_{5a}(n) = \frac{\{\}, True, (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + 1 + n)}{210 \overset{\tau}{\Rightarrow} 100}$$

$$WI_6(n) = \frac{\{\}, True, (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + 1 + n)}{220 \overset{\tau}{\Rightarrow} 100}$$

$$WI_{6a}(n) = \frac{\{\}, True, (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec} + 1 + n, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + 1 + n)}{220 \overset{\tau}{\Rightarrow} 210}$$

$$WI_{456*}(n) = \frac{\{\}, True, (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \overset{\tau}{\Rightarrow} 100}$$

$$WI_{564*}(n) = \frac{\{\}, True, (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + 1 + n, \texttt{m\_ec} \leftarrow \texttt{s\_ec} + 1 + n)}{210 \overset{\tau}{\Rightarrow} 210}$$

$$WI_{645*}(n) = \frac{\{\}, True, (\texttt{s\_ec} \leftarrow \texttt{s\_ec} + 1 + n)}{220 \overset{\tau}{\Rightarrow} 220}$$

$$WI_7(n) = \frac{\{\}, True, (\texttt{r\_msg} \leftarrow \texttt{s\_msg}, \texttt{r\_ec} \leftarrow \texttt{s\_ec} + n)}{210 \overset{\tau}{\Rightarrow} 201}$$

$$WI_{7a}(n) = \frac{\{\}, True, (\texttt{r\_msg} \leftarrow \texttt{s\_msg}, \texttt{r\_ec} \leftarrow \texttt{m\_ec} + n + 1)}{220 \stackrel{\tau}{\Rightarrow} 201}$$

$$WI_{7b}(n) = \frac{\{\}, True, (\texttt{r\_msg} \leftarrow \texttt{m\_msg}, \texttt{r\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \stackrel{\tau}{\Rightarrow} 201}$$

$$WI_{8} = \frac{\{\texttt{Q} \mapsto \texttt{q-recv(r1\_msg,r1\_ec)}\}, True, ()}{201 \stackrel{\texttt{out(r1\_msg,r1\_ec)}}{\Longrightarrow} \{202, 000\}}$$

$$WI_{8a}(n) = \frac{\{\texttt{Q} \mapsto \texttt{q-recv(m\_msg,m\_ec} + n)\}, True, ()}{210 \stackrel{\texttt{out(m\_msg,m\_ec}+n\texttt{)}}{\Longrightarrow} \{202, 000\}}$$

$$WI_{8b}(n) = \frac{\{\texttt{Q} \mapsto \texttt{q-recv(s\_msg,s\_ec} + n + 1)\}, True, ()}{220 \stackrel{\texttt{out(s\_msg,m\_ec}+n+1\texttt{)}}{\Longrightarrow} \{202, 000\}}$$

$$WI_{8c}(n) = \frac{\{\texttt{Q} \mapsto \texttt{q-recv(s\_msg,s\_ec} + n)\}, True, ()}{100 \stackrel{\texttt{out(s\_msg,s\_ec}+n\texttt{)}}{\Longrightarrow} \{202, 000\}}$$

Then for all τ transitions above we have a similar WOT that include a non-τ move from an external action of P or Q, like for example:

$$WI_4P(n) = \frac{\{\texttt{P} \mapsto \texttt{p-a}\}, [\forall \texttt{m.p-a} \neq \texttt{p-send(m)}], (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec} + n, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \stackrel{\texttt{p-a}}{\Longrightarrow} 210}$$

and

$$WI_4Q(n) = \frac{\{\texttt{Q} \mapsto \texttt{q-b}\}, [\forall \texttt{m,ec.q-b} \neq \texttt{q-recv(m,ec)}], (\texttt{m\_msg} \leftarrow \texttt{s\_msg}, \texttt{m\_ec} \leftarrow \texttt{s\_ec} + n, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \stackrel{\texttt{q-b}}{\Longrightarrow} 210}$$

but also e.g.:

$$WI_{456*}P(n) = \frac{\{\texttt{P} \mapsto \texttt{p-a}\}, [\forall \texttt{m.p-a} \neq \texttt{p-send(m)}], (\texttt{s\_msg} \leftarrow \texttt{s\_msg}, \texttt{s\_ec} \leftarrow \texttt{s\_ec} + n)}{100 \stackrel{\texttt{p-a}}{\Longrightarrow} 100}$$

The following table give a summary of WOTs, when sharing their names as much as possible.

That makes a total of 73 (meta-)WOTs in the Figure 5.

meta-WOT name	Pairs of source states and target states	# of WOTs
WI_1 WI_2 WI_τ	$\{(s,s)\|s \in$ `States of WOA`$\} \cup \{(202,000)\}$	21
$WI_3(n)$	{(202,100),(000,100)}	2
$WI_{3a}(n)$	{(202,210),(000,210)}	2
$WI_{3b}(n)$	{(202,220),(000,220)}	2
$WI_{3c}(n)$	{(202,201),(000,201)}	2
$WI_4(n)$ $WI_4P(n)$ $WI_4Q(n)$	{(100,210)}	3
$WI_{4a}(n)$ $WI_4aP(n)$ $WI_4aQ(n)$	{(100,220)}	3
$WI_{456*}(n)$ $WI_{456*}P(n)$ $WI_{456*}Q(n)$	{(100,100)}	3
$WI_5(n)$ $WI_5P(n)$ $WI_5Q(n)$	{(210,220)}	3
$WI_{5a}(n)$ $WI_{5a}P(n)$ $WI_{5a}Q(n)$	{(210,100)}	3
$WI_{564*}(n)$ $WI_{564*}P(n)$ $WI_{564*}Q(n)$	{(210,210)}	3
$WI_6(n)$ $WI_6P(n)$ $WI_6Q(n)$	{(220,100)}	3
$WI_{6a}(n)$ $WI_{6a}P(n)$ $WI_{6a}Q(n)$	{(220,210)}	3
$WI_{645*}(n)$ $WI_{645*}P(n)$ $WI_{645*}Q(n)$	{(220,220)}	3
$WI_7(n)$ $WI_7P(n)$ $WI_7Q(n)$	{(210,201)}	3
$WI_{7a}(n)$ $WI_{7a}P(n)$ $WI_{7a}Q(n)$	{(220,201)}	3
$WI_{7b}(n)$ $WI_{7b}P(n)P$ $WI_{7b}Q(n)$	{(100,201)}	3
$WI_8(n)$	{(201,202),(201,000)}	2
$WI_{8a}(n)$	{(210,202),(210,000)}	2
$WI_{8b}(n)$	{(220,202),(220,000)}	2
$WI_{8c}(n)$	{(100,202),(100,000)}	2

Table 6: All meta-WOT in the Implementation WOA

A.1 Details of the Bisimulation Checking

We recall here the relation $\mathcal{R}$ that is the candidate for our meta FH-Bisimulation relation in Table 7.

Considering the first triple <b0, 000, True>, we will prove the following properties, in which $OT << mWOT$ means that the (strong) open transition OT is covered, in the sense of Definition 3.5 by the meta-WOT $mWOT$:

Spec state	Impl state	Predicate
b0	000	True
b0	202	True
b1	100	`b_msg = s_msg` $\wedge$ `b_ec = s_ec`
b1	210	`b_msg = m_msg` $\wedge$ `b_ec = m_ec`
b1	220	`b_msg = s_msg` $\wedge$ `b_ec = s_ec`
b1	201	`b_msg = r_msg` $\wedge$ `b_ec = r_ec`

Table 7: Bisimulation relation Triples of running example

$$
\begin{aligned}
WS_1 << WI_1(\emptyset) \qquad & WI_1 << WS_1(\emptyset) \\
WS_2 << WI_2(\emptyset) \qquad & WI_2 << WS_2(\emptyset) \\
\forall n \geqslant 0, n' \geqslant 0.\, & WS_3(n) << WI_3(n') \\
\forall n \geqslant 0, n' \geqslant 0.\, & WI_3(n) << WS_3(n') \\
\forall n \geqslant 0, n' \geqslant 0.\, & WI_3a(n) << WS_3(n') \\
\forall n \geqslant 0, n' \geqslant 0.\, & WI_3b(n) << WS_3(n') \\
\forall n \geqslant 0, n' \geqslant 0.\, & WI_3c(n) << WS_3(n')
\end{aligned}
$$

B Alt-Meta FH-Bisimulation

In previous section, we introduce the meta-Variable by Self-Loop transitions which leads to infinity loops. But, in another situation, the meta-Variable can be used by user design. In other words, meta-Variable can be applied directly in strong OA. This section is a first try to extend the "meta variable" idea to be used directly by application architects as a "early time" specification formalism. For this we define meta transitions and meta automata that can be used from scratch, not only as the result of computing weak transitions; then we give a direct definition of a (strong) FH-Bisimulation for such systems. The study of the relations between these notions and those developed previously in the core of paper is still work in progress.

Definition B.1 (Meta-OT and meta-OA) *A Meta-OA, the abbreviation of meta (strong) open automaton, is a structure $A = \langle\!\langle J, S, s_0, V, M, \mathcal{MT} \rangle\!\rangle$ where:*

- *J, S, s_0 is similar to an open automaton.*
- *V is the set of variables, M is the set of meta-Variables, with $V \cup M = \emptyset$. We denote $V = vars(A)$, $M = mvars(A)$.*
- *$\mathcal{MT}$ is a set of meta open transitions which is similar to the open transition but needs a set of meta-Variables to express its semantic meaning.*

A meta open transition, (meta-OT or mOT for short), is a structure similar to open transitions, of the form:

$$meta\text{-}OT(M') = \frac{\gamma_j^{j \in J'}, Pred, Post}{s \xrightarrow{\alpha} s'}$$

This is similar to Definition 2.4, but includes a set of meta-Variables $M' \subseteq M$ that can be used in all the expressions γ, Pred, Post, and α. Each $mv \in M$ has its domain, $dom(mv)$. Notice that when the set of meta-Variables is $\emptyset$, or each meta-Variable has one single legal valuation, then the meta-OT is a normal OT.

Then we need some construction rules for meta-OT to build meta-WOT.

Definition B.2 (Building meta-WOT from meta-OT) *There are three rules for building meta-WOT from meta-OT.*

$$mot_\tau(\emptyset) = \frac{\emptyset, True, \mathrm{Id}(V)}{s \stackrel{\tau}{\Rightarrow} s} \in \mathcal{MT} \qquad \textbf{\textit{MWT1}}$$

and

$$\frac{mot_1(M') = \overset{\overline{\beta},\, Pred,\, Post}{\cdots\cdots\cdots\cdots\cdots\cdots} \in \mathcal{MT}}{mwot_1(M') = \overset{(\overline{\beta})^{\nabla},\, Pred,\, Post}{\cdots\cdots\cdots\cdots\cdots\cdots} \in \mathcal{MWT}} \quad \textbf{MWT2}$$

$$s \xrightarrow{\alpha} s' \qquad s \overset{\alpha}{\Rightarrow} s'$$

and

$$\frac{\begin{array}{c} mot_1(M_1') = \overset{\overline{\gamma_1},\, Pred_1,\, Post_1}{\cdots\cdots\cdots\cdots\cdots\cdots} \in \mathcal{MWT} \quad mot_2(\sigma_V(M_2')) = \overset{\overline{\gamma_2},\, Pred_2,\, Post_2}{\cdots\cdots\cdots\cdots\cdots\cdots} \in \mathcal{MWT} \\ s \overset{\tau}{\Rightarrow} s_1 \qquad\qquad\qquad\qquad s_1 \overset{\alpha}{\Rightarrow} s_2 \\ mot_3(\sigma_V(M_3')) = \overset{\overline{\gamma_3},\, Pred_3,\, Post_3}{\cdots\cdots\cdots\cdots\cdots\cdots} \in \mathcal{MWT} \\ s_2 \overset{\tau}{\Rightarrow} s' \\ Pred = Pred_1 \wedge Pred_2\{\!|Post_1|\!\} \wedge Pred_3\{\!|Post_2 \otimes Post_1|\!\} \quad \overline{\gamma} = \overline{\gamma_1} \cup \overline{\gamma_2}\{\!|Post_1|\!\} \cup \overline{\gamma_3}\{\!|Post_2 \otimes Post_1|\!\} \\ \alpha' = \alpha\{\!|Post_1|\!\} \qquad M_1' \cup \sigma_V(M_2') = \emptyset \wedge M_1' \cup \sigma_V(M_3') = \emptyset \wedge \sigma_V(M_2') \cup \sigma_V(M_3') = \emptyset \\ M' = M_1' \cup \sigma_V(M_2') \cup \sigma_V(M_3') \end{array}}{mwot(M') = \overset{\overline{\gamma},\, Pred,\, Post_3 \otimes Post_2 \otimes Post_1}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots} \in \mathcal{MWT}} \quad \textbf{MWT3}$$

$$s \overset{\alpha'}{\Rightarrow} s'$$

*Meta-Variable with quantifier is local variable, σ_V renames them to make sure that meta-Variables (especially these with quantifier) in the result of **MWT3** has unique name. And new meta-Variables can be introduced by Self-Loop transitions also.*

When meta-OT's meta-Variables set is $\emptyset$, it's OK to omit the brackets and terms in them.

With Definition B.2, we can derive meta-WOA from meta-OA. Then it's necessary to define a kind of new meta FH-Bisimulation for meta-OAs.

Definition B.3 (Alt-meta FH-Bisimulation) *Let OAs $moa_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, M_1, \mathcal{MT}_1 \rangle\!\rangle$ and $moa_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, M_2, \mathcal{MT}_2 \rangle\!\rangle$ be two meta strong open automata, from which are derived meta-WOAs: $mwoa_1 = \langle\!\langle J, \mathcal{S}_1, s_0, V_1, M_1, \mathcal{MWT}_1 \rangle\!\rangle$ and $mwoa_2 = \langle\!\langle J, \mathcal{S}_2, t_0, V_2, M_2, \mathcal{MWT}_2 \rangle\!\rangle$.*

A TripleSet $\mathcal{R}$ is a meta-FH Bisimulation iff for any states $s \in \mathcal{S}_1$ and $t \in \mathcal{S}_2$, $(s, t|Pred_{s,t}) \in \mathcal{R}$, we have the following:

- *For any meta open transition $mot(M_1') \in \mathcal{MT}_1$:*
  $\overset{\gamma_j^{j \in J'},\, Pred_{mwot},\, Post_{mwot}}{\cdots\cdots\cdots\cdots\cdots\cdots}, M_1' \subseteq M_1$
  $s \overset{\alpha}{\Rightarrow} s'$

  there exist a set of meta weak open transitions $mwot_x^{x \in X}(M_2') \subseteq \mathcal{MWT}_2$:
  $\overset{\gamma_{jx}^{j \in J_x},\, Pred_{mwot_x},\, Post_{mwot_x}}{\cdots\cdots\cdots\cdots\cdots\cdots}, M_2' \subseteq M_2$
  $t \overset{\alpha_x}{\Rightarrow} t_x$

  such that $\forall x, J' = J_x, Pred_{s',t_x}.(s', t_x|Pred_{s',t_x}) \in \mathcal{R}$; and

  $\forall\ otvars(mwot).\Big\{ Pred_{s,t} \wedge Pred_{mwot} \implies$

$$\bigvee_{x\in X}\Big[\exists\ otvars(mwot_x).$$
$$\Big(\forall j.\gamma_j = \gamma_{jx} \wedge Pred_{mwot_x} \wedge \alpha{=}\alpha_x \wedge Pred_{s',t_x}\{\!\{Post_{mwot} \uplus Post_{mwot_x}\}\!\}\Big)\Big]\Big\}$$

- *and symmetrically any meta open transition from t in $\mathcal{MT}_2$ can be covered by a set of transitions from s in $\mathcal{MWT}_1$.*

Two OAs are Alt-meta FH-Bisimilar w.r.t. some initial condition $Pred_0$ if there exists a meta FH-Bisimulation relation between their associated automata, and their initial states are in the relation, i.e., the predicate associated with the relation between the initial states is $Pred_0$.

Remark that the triple predicates (and here $Pred_0$) can only include standard automata variables, not meta-variables.

B.1 Upgraded Checking Algorithm

The checking algorithm for weak FH-Bisimulation only fits the OAs and corresponding WOTs, without meta-Variables. Moreover, it only looks for one WOT, which can cover the current OT. However, in normal cases (according to the definition of bisimulation), a (meta-)OT could be covered by a set of (meta-)WOTs, where each (meta-)WOT only covers a part of semantic of the (meta-)OT. Just like the $Amot_1(\{x\})$ is covered by a meta-WOT set S in section C.1.

Considering these two situations, we have designed an upgraded checking algorithm for Alt-meta FH-Bisimulation. It is also bounded but slightly modified compared to the original algorithm.

The main body of the algorithm is $CHECKTRIPLES2$. When M_1 and M_2 are $\emptyset$, it is an upgraded checking algorithm for weak FH-Bisimulation.

Algorithm 5 Checking Alt-meta FH-Bisimulation between Two meta-OAs by gathering WOTs

input: mOA_1 and mOA_2 two meta open automata, where $mOA_1 = \langle\!\langle J_1, \mathcal{S}_1, init_1, V_1, M_1, \mathcal{T}_1 \rangle\!\rangle$ and $mOA_2 = \langle\!\langle J_2, \mathcal{S}_2, init_2, V_2, M_2, \mathcal{T}_2 \rangle\!\rangle$. $\mathcal{R}$ is a set of Triples, where $\mathcal{R} = \{(s, t|Pred_{s,t}) | s \in \mathcal{S}_1, t \in \mathcal{S}_2\}$

output: A variable with two possible values: $True$ means the input OAs have relation R; $Failed$ means the algorithm failed to decide the relation.

```
1  function CHECKTRIPLES2(mOA_1, mOA_2, R)
2      mwoa_2 ← apply MWT2 on mOA_2
3      for each Triple (s, t|Pred_{s,t}) in R
4          for each Open Transition mot in mOA_1 from state s to s'
5              mWOTSet ← ∅
6              for each Triple with first element as s', says
7              Triple' = (s', t'|Pred_{s',t'})
8                  mwot_τ ← apply MWT1 on t
9                  TMPmWOTSet ← ∅
10                 GATHERWOTS(mwoa_2, mot, Triple, Triple', mwot_τ, TMPmWOTSet)
11                 mWOTSet ← mWOTSet ∪ TMPmWOTSet
12             end for
13             negateOb ← generate negation of proof obligation (mot,
                   mWOTSet and corresponding Triples)
14             use SMT-solver to check negateOb's satisfiability
15             if SMT-solver returns UNSAT then
16                 continue
17             else then
18                 return Failed
19             end if
20         end for
21     end for
22     return True
23 end function
```

In this function, we traverse all the Triples and build corresponding meta-WOTs from one meta-OA on demand and check whether all of them meet the definition of Alt-meta FH-Bisimulation. After this, we build meta-WOTs from the other meta-OA for the symmetrical property and process it. It is the same for both. Thus we only show the first direction.

We have input with two meta-OAs, mOA_1 and mOA_2, and Triples set $\mathcal{R}$. First, we build a new meta-WOA, $mwoa_2$, from mOA_2 only to apply **MWT2**. Let $(s, t|Pred_{s,t})$ become the Triple we are checking now. For each meta-OT starting from s (say, mot from s to s'), we find out

these Triples with their first item as s'. From each of these Triples, say, $(s', t'|Pred_{s',t'})$, we try to find all the meta-WOTs from t to t' by the bounded search algorithm by calling $GATHERWOT$.

$GATHERWOT$ is a bounded DFS(BFS) algorithm with a bound of search depth. Once we get one expected meta-WOT (from t to t'), we put it into the set, the last parameter, which is an empty set at the first call.

After $GATHERWOT$ is finished, we get all the meta-WOTs from t to t' within the bound. Then we go for next Triple $(s', t''|Pred_{s',t''})$, for gathering meta-WOTs from t to t''. After all Triples whose first item is s' are traversed, we construct a corresponding (negation) proof obligation (*mot* with s set of meta-WOTs) to check the satisfiability by SMT engine, Z3.

When Z3 returns $UNSAT$ (it means the proof obligation is a tautology), that means our current meta-OT (variable *mot*) is covered, and we continue this process for all other meta-OTs from s. Otherwise, it returns $Failed$, which means the algorithm failed to judge the relation.

If all of the meta-OTs (from s) are covered, then the conclusion is that $(s, t|Pred_{s,t})$ fits the definition of Alt-meta FH-Bisimulation, and the first loop goes for the next Triple.

If all the Triples pass this check, it means all the Triples in $\mathcal{R}$ meet the requirements; thus, we can say $\mathcal{R}$ is an Alt-meta FH-Bisimulation between OA_1 and OA_2 and return true.

For $GATHERWOTS$, there are two kinds of searching styles, DFS and BFS. We only show the pseudocode of Bounded DFS in the paper. It simply searches the automaton by building meta-WOTs and gathers the ones which are from t to t'.

Algorithm 6 Gather mWOTs on demand to check $\mathcal{R}$ with target mOT

input: *mwoa* is a meta-WOA, *mot* (from s to s') is the meta-OT needs to be covered , $Triple$ $((s, t|Pred_{s,t}))$ is the Triple in $\mathcal{R}$ which need to be checked, $Triple'$ $((s', t'|Pred_{s',t'}))$ is the corresponding Triple of *ot*, *wot*(from t to t_{mid} with action α') is a WOT, $mWOTSet$ is a set collection for expected meta-WOTs.

output: A set collection of expected meta-WOTs.

```
1  function GATHERWOTS(WOA, OT, Triple, Triple', wot, mWOTSet)
2      if length(mwot) <= LIMITS then
3          for each mwot' in mwoa from t_mid, says to t'_x
4              α'' ← action of mwot'
5              if (α' is non−observable) or (α'' is non−observable) then
6                  newmwot ← Combine(mwot, mwot')
7                  if t'_x = t' then
8                      mWOTSet ← mWOTSet ∪ {newmwot}
9                  end if
10             end if
11         end for
12     end if
13 end function
```

This algorithm's possible disadvantage is that it may generate a giant proof obligation for Z3, which may make Z3 collapse!

C Examples

In this section, we introduce some other exciting examples of Weak FH-Bisimulation (meta FH-Bisimulation) for better understanding. By lack of time, we have not yet checked the properties and equivalences of these examples.

C.1 Integer Printer

There are two kinds of integer printer, $A(x)$ and $B(y)$, with different structure which can print any positive integer. $A(x)$ and $B(y)$ are strong automata with meta-Variable x and y respectively.

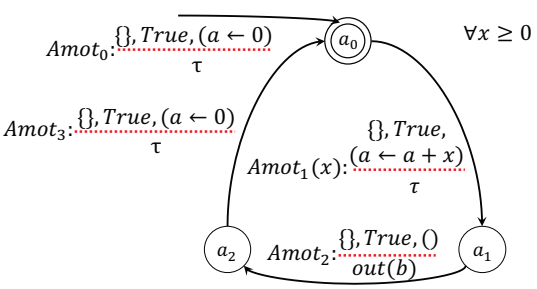


Figure 16: Integer Printer A(x)

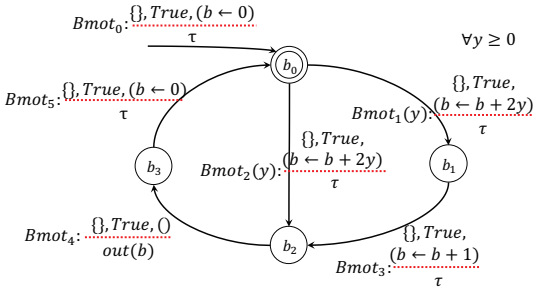


Figure 17: Integer Printer B(y)

And the Predicates in Table 8.

A(x)	B(y)	Predicates
$<a_0>$	$<b_0>$	$a = b = 0$
$<a_1>$	$<b_1>$	$odd(a) \wedge a = b + 1$
$<a_1>$	$<b_2>$	$a = b$
$<a_2>$	$<b_3>$	$True$

Table 8: Predicates of A(x) and B(y)

It's easy to see that $Amot_0$ and $Bmot_0$ are Similar; $Amot_2$ and $Bmot_4$ are Similar; $Amot_3$ and $Bmot_5$ are Similar.

We have a meta-WOT set $S = \left\{mwot_2(y), mwot_{13}(y)\right\}$ from $B(y)$, where $mwot_2(y)$ is built by **MWT2** from $Bmot_2(y)$ and $mwot_{13}(y)$ is built by **MWT3** from meta-OT sequence $Bmot_1(y); Bmot_3$:

$$mwot_2(\{y\}) = \frac{\{\}, True, (b \leftarrow b + 2y)}{b_0 \stackrel{\tau}{\Rightarrow} b_2} \qquad mwot_{13}(\{y\}) = \frac{\{\}, True, (b \leftarrow b + 2y + 1)}{b_0 \stackrel{\tau}{\Rightarrow} b_2}$$

The proof obligation that $Amot_1(x)$ is covered by S is:

$$\forall x \geqslant 0. \Big\{ Pred_{a_0,b_0} \wedge True \implies \Big[\exists y \geqslant 0. \Big(\tau = \tau \wedge Pred_{a_1,b_2} \{\!\![a \leftarrow a + x] \uplus [b \leftarrow b + 2y]\!\!\} \Big) \vee \Big(\tau = \tau \wedge Pred_{a_1,b_2} \{\!\![a \leftarrow a + x] \uplus [b \leftarrow b + 2y + 1]\!\!\} \Big) \Big] \Big\}$$

Instantiated as:

$$\forall x \geqslant 0.\Big\{a = b = 0 \implies \big[\exists y \geqslant 0.(\tau = \tau \wedge [a + x = b + 2y]) \vee (\tau = \tau \wedge [a + x = b + 2y + 1])\big]\Big\}$$

That is a tautology.

And $Bmot_1(y)$ and $Bmot_2(y)$ are covered by the meta-WOT built from $Amot_1(y)$. A interesting part is that $Bmot_3$ is covered by Self-Loop τ transition built by **MWT1** on state a_1. And its proof obligation is:

$$\forall x \geqslant 0.\Big\{(odd(a) \wedge a = b + 1 \implies \big[\exists y \geqslant 0.\tau = \tau \wedge [a = b]\{b \leftarrow b + 1\}\big]\Big\}$$

When a is odd, then the proof obligation can be simplified as:

$$\forall x \geqslant 0.\Big\{a = b + 1 \implies \big[\exists y \geqslant 0.a = b + 1\big]\Big\}$$

It's tautology.

When a is even, then the proof obligation can be simplified as:

$$\forall x \geqslant 0.\Big\{False \implies \big[\exists y \geqslant 0.a = b + 1\big]\Big\}$$

$False$ implies anything. So, it's tautology. As conclusion, $A(x)$ and $B(y)$ are Alt-meta FH-Bisimulation.

We derive meta-WOT from $A(x)$ as Figure 18.

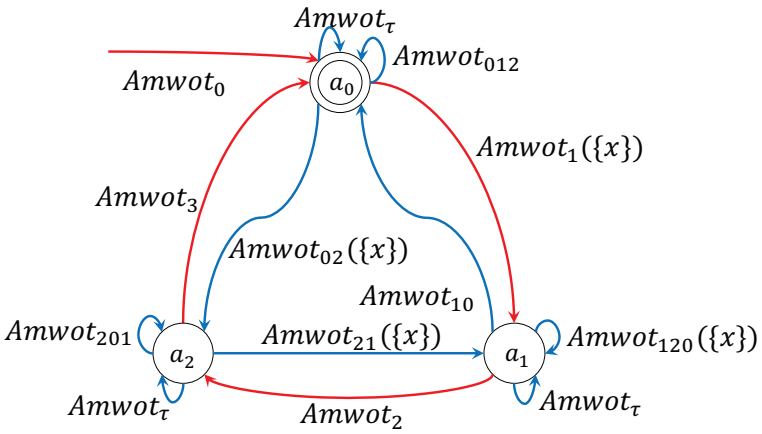


Figure 18: Integer Printer A(x) of meta-WOA

We have $Amwot_i, \forall i \in \{0,1,2,3\}$ come from $Amot_i, \forall i \in \{0,1,2,3\}$ (all the blue lines in Figure 18); $Amwot_\tau = \overset{\{\},True,()}{\underset{s_0 \overset{\tau}{\Rightarrow} s_0}{\cdots\cdots\cdots\cdots}}, \forall s_0 \in \{a_0, a_1, a_2\}$;

$$Amwot_{10} = \frac{\{\}, True, (a \leftarrow 0)}{a_1 \xrightarrow{out(a)} a_0} \qquad Amwot_{012} = \frac{\{\}, True, (a \leftarrow 0)}{a_0 \xrightarrow{out(a)} a_0}$$

$$Amwot_{02}(\{x\}) = \frac{\{\}, True, (a \leftarrow 0 + x)}{a_0 \xrightarrow{out(a)} a_2} \qquad Amwot_{120}(\{x\}) = \frac{\{\}, True, (a \leftarrow 0 + x)}{a_1 \xrightarrow{out(a)} a_1}$$

$$Amwot_{21}(\{x\}) = \frac{\{\}, True, (a \leftarrow 0 + x)}{a_2 \xRightarrow{\tau} a_1} \qquad Amwot_{201}(\{x\}) = \frac{\{\}, True, (a \leftarrow 0 + x)}{a_2 \xrightarrow{out(a)} a_2}$$

This is a example where the Self-Loop Transition does not introduce a meta-Variable.

Spec	Impl	Predicates
<b0>	<c0r0a0>	b_m=m ∧ b_coin=coin ∧ b_n=n ∧ b_a=a
<b0>	<c0ra0>	b_m=m ∧ b_coin=coin ∧ b_n=n ∧ b_a=a
<b1>	<c1r1a0>	b_m=m ∧ b_coin=coin ∧ b_n=n ∧ b_a=a
<b1>	<c1r0a0>	b_m=m ∧ b_coin=coin ∧ b_n=n ∧ b_a=a ∧ b_p=2

Table 9: Bisimulation Relation Triples of Vendor Machine

C.2 Coffee Machine

The idea of coffee vendors comes from paper [4]. It describes how a coffee vending machine works. The differences are that we have Holes, predicates, and assignments, not only action labels. So, with a little extension, we design the new specification open automaton for the vendor machine, in Figure 19. It has two states and five actions on them. The vendor machine only has two participants, the customer named Hole P and the manager who cleans the coin box and supplies coffee named Hole M.

Customers can start the transaction, deposit coins, confirm the transaction, and withdraw the extra coins. The manager can manage the machine, supplying new coffee materials. If some customers forget to withdraw, the manager can reset the machine.

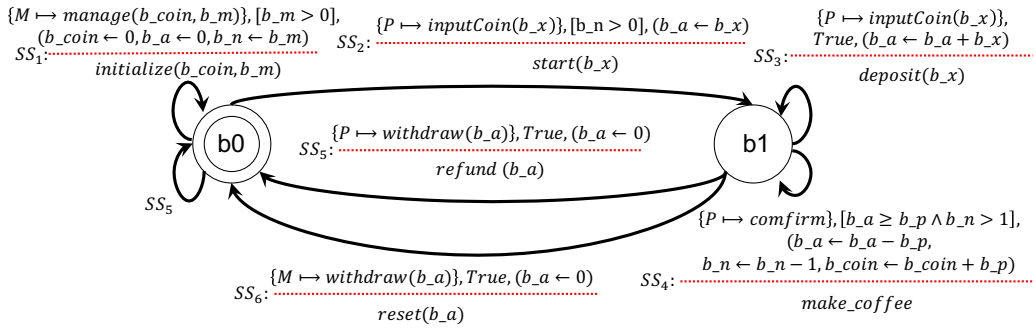


Figure 19: Specifications OA of Coffee Vendors

Then we design the details of the coffee vending machine system in the form of pNet, Figure 20. It has two levels. The top-level organizes the cooperation of the system and customers. The bottom-level has three leaves automaton, Account, Counter, Money Box, and one Hole M, the manager.

The Account is in charge of the account of current transactions. And it only accepts coins with a face value of 1 Euro and 2 Euro. The price of coffee is 2 Euro. The Counter counts the stock of the countable coffee materials, maybe the capsules. The Money Box records the incomes since the last time the manager clean the coin box. The manager will simultaneously complete the two actions, from the system's view, clean the coin box, and supply the coffee materials (capsules) to the machine's maximum volume (b_m and m).

Using the operational semantics of pNets, we get the implementation open automaton from the pNet. In this OA, Figure 21, we have a non-observable action from synchronized vectors, runout. We expect a Weak FH-Bisimulation equivalence between specification and implementation, with the relation Tripleset defined as Table 9.

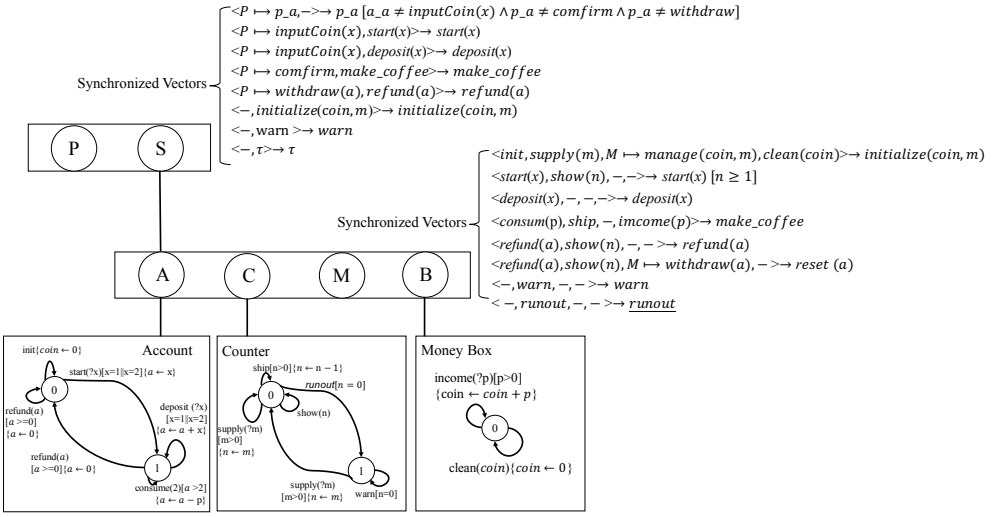


Figure 20: pNet Implementation of Coffee Vendors

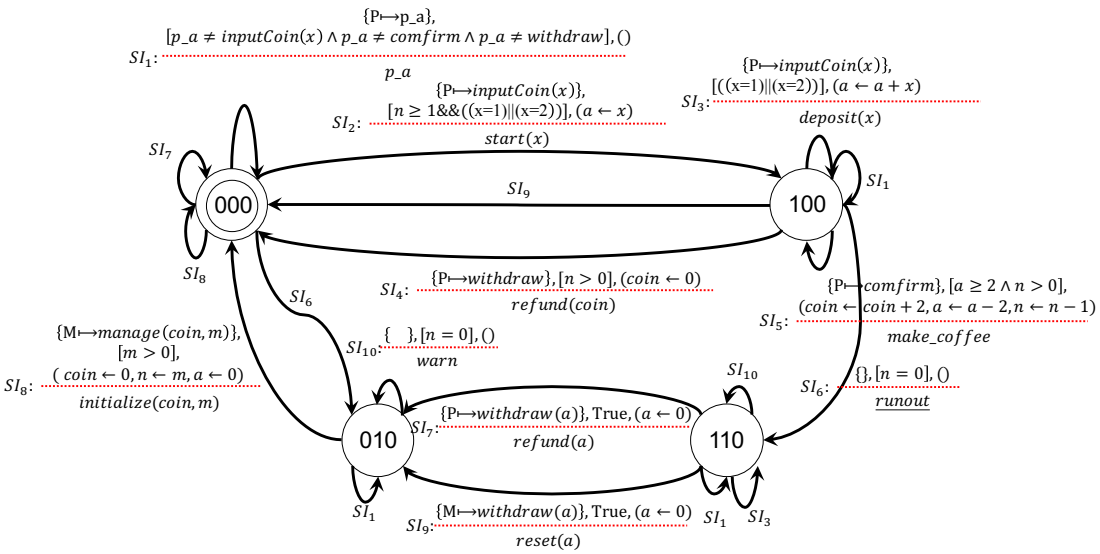


Figure 21: Implementations OA of Coffee Vendors

C.3 Traffic Lights

This example is about traffic lights. Traffic lights are an appropriate use-case for real-time systems [**cohen2015systemverilog**]. It controls a traffic light at the intersection of a busy highway and a farm road. We simplify it as a normal crossroad and abstract the timer as a Hole in pNet.

There are three kinds of lights in one traffic light, Green, Red, and Yellow. Here, the only participant is a pedestrian who should wait when the Red Light on; walk through the crossroads when Green Light on; Pay attention when the Yellow Light is illuminating. It is just like the specification in Figure 22. Nevertheless, the time interval for each light is set by prior knowledge: 3s for Yellow Light, 17s for Green Light, and 20s for Red Light. With two sets of these traffic light systems controlling two directions, we can control a crossroad.

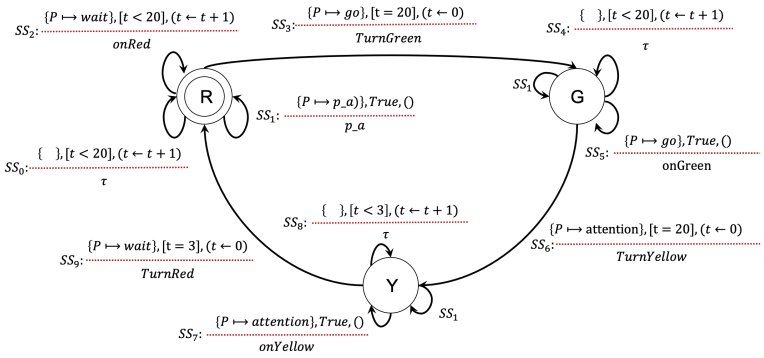


Figure 22: Specification OA of a Traffic Light

Then we design the details of the traffic lights system in the form of pNet, Figure 23. It has two levels. On the top level, it describes the cooperation between the traffic light system and pedestrians.

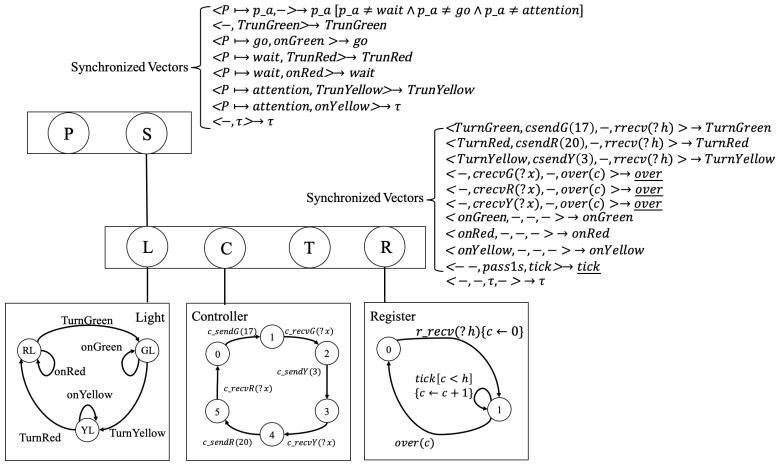


Figure 23: pNet Implementations of Traffic Light

On the bottom level, it describes how the several components, sub-systems, work with each other. The 'Light' only control which light should be on, and it will turn on another after turn off the current light; the 'Controller' is in charge of communications between the other sub-systems by sending time thresholds and receive data. The 'Register' is a counter which starts to count after receiving a time threshold. Furthermore, there is an ideal timer, Hole 'T', which has only one action, telling it has just passed 1 second. With the help of the Hole 'T', the register can count the seconds correctly.

Using the operational semantics of pNet, we get the implementation open automaton from the pNet. In this OA, Figure 24, the most interesting part is that the work of register (timer) is independent because the environment cannot observe it. The pedestrian does actions only by the signal of light.

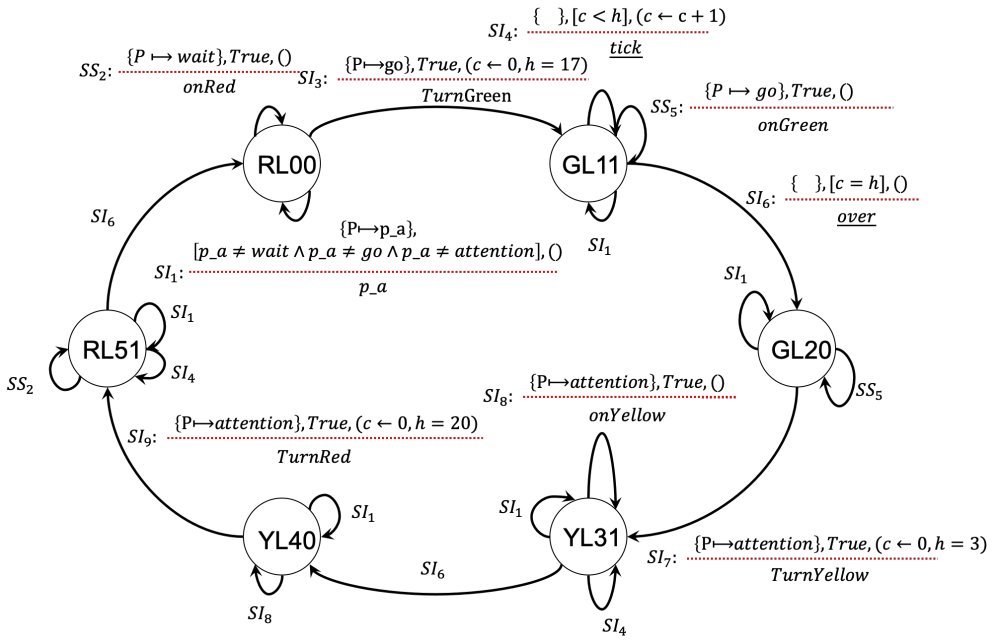


Figure 24: Implementation OA of Traffic Light

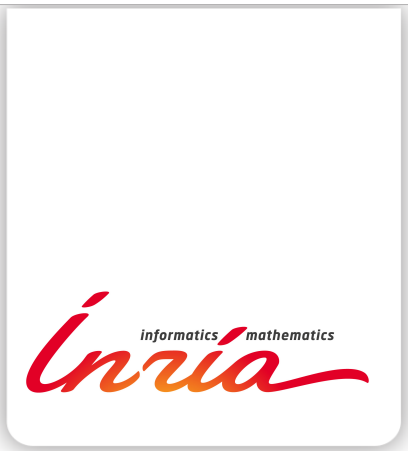
The relation Triples are given like this:

Spec	Impl	Predicates
<R>	<RL51>	$c = t \wedge h = 20$
<R>	<RL00>	$c = t \wedge c = h$
<G>	<GL11>	$c = t \wedge h = 17$
<G>	<GL20>	$c = t \wedge c = h$
<Y>	<YL31>	$c = t \wedge h = 3$
<Y>	<YL40>	$c = t \wedge c = h$

Table 10: Bisimulation Relation Triples of Traffic Lights

D Acknowledgement

This work is the central achievement of Biyang Wang master studentship under the supervision of M. Eric Madelaine. Eric Madelaine is supported by Inria (France), and by the Associated Team FM4IoT between Inria and ECNU Shanghai. Min Zhang is partially supported by the NSFC Project (No. 61672012). BiYang Wang is partially supported by both of above institutions and by a studentship grant from Université Côte d'Azur DS4H. BiYang Wang would like to thank Xudong Qin, Zechen Hou and Rabéa Ameur-Boulifa for their help to fully understand the previous works (Open Automata and StrFH-Bisimulation theories, etc.), the VerCors verification platform, and their implementation of previous algorithms.



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399